Transition-Based Parsing for Large-Scale Head-Driven Phrase Structure Grammars

Doctoral Dissertation by

Gisle Ytrestøl

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo

Submitted for the degree of Philosophiae Doctor

April 2012

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1211

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AIT Oslo AS.

Produced in co-operation with Akademika publishing.
The thesis is produced by Unipub merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

Abstract

Deterministic, transition-based parsing has seen a surge of interest over the recent decade, with research efforts targeting Dependency Grammar, Context-Free Grammar, Head-Driven Phrase Structure Grammar (HPSG), and Combinatory Categorial Grammar. Previous work, however, has not applied the transition-based approach to parsing with hand-crafted, large-scale unification-based grammars.

Basing our studies on the English Resource Grammar (ERG), we evaluate the feasibility of transferring strategies and methods from other transition-based approaches to a semantically 'deep', hand-crafted HPSG. Our parsing platform, dubbed CuteForce, constitutes a pipeline which assumes pretokenized sentences, and produces syntacto-semantic representations in accordance with the ERG framework. The components in this pipeline include a preprocessing and supertagging stage and a transition-based parsing stage where both deterministic and near-deterministic strategies are evaluated. We evaluate the supertagger in isolation, and compare our overall parsing results to other ERG parsers. This allows us to assess the trade-offs a transition-based parsing approach for large-scale HPSGs may have in terms of parser precision, robustness and efficiency, compared to 'classic' parsing approaches.

Both the preprocessing stage and the transition-based parser rely on large amounts of training data. To ensure that we had sufficient linguistic resources for our data-driven platform, the first part of the project was committed to extracting a corpus from Wikipedia, and convert this data to a gold standard treebank (WeScience Treebank) and a 'silver standard' parsed corpus (WikiWoods). Utilizing Wikipedia as a linguistic resource has received increased attention, and we expect that the methodology for corpus acquisition presented in this thesis could also prove useful to other research initiatives.

We find that large amounts of 'silver standard' training data allows us to train a supertagger that reaches a previously unmatched level of supertagging accuracy for the ERG. Further, our evaluation shows that although the transition-based parser does not obtain state-of-the-art accuracy, it still reaches a high level of accuracy, coupled with much higher parsing efficiency than other parsers based on the same grammar, making it a suitable choice amongst others when speed has high priority.

Acknowledgements

After four years of working on this PhD project, there are many people to whom I owe a debt of gratitude. First I would like to thank my advisor Stephan Oepen. His relentless enthusiasm, support, assistance and cryptic scribblings have followed me and been my most important inspiration throughout the PhD project. Further, I would like to extend my gratitude to my two co-supervisors. Jan Tore Lønning has given me invaluable feedback during this PhD project, always prepared to share his vast insights and expertise in areas such as parsing and mathematical aspects. Joakim Nivre most generously agreed to take on the role as co-supervisor during the middle of this project, and he has, with his in-depth knowledge of (amongst other things) transition-based parsing, been an invaluable resource for me in the project. With this distinguished team of (co-)advisers, there were no external factors to blame if I did not successfully pull this project through.

In this context, I would also like to thank all of my colleagues at the Language Technology Group at the University of Oslo. From this group, I would particularly like to emphasize the support from Rebecca Dridan, who I have been asking about every aspect of supertagging, and who also helped me with many a linguistic challenge that arose for a non-native speaker attempting to write a thesis in English. Additionally, I want to thank Erik Velldal, who has always been prepared to help me out with anything ranging from machine learning questions to caffeine deficit.

The DELPH-IN community has been an outstanding resource, and in this context I would especially like to express my gratitude to Dan Flickinger, who has been involved in all aspects regarding the linguistic resources that have been applied or developed in this project. I would also like to thank Ivan Sag, who, and together with Flickinger made my stay as a visiting scholar at CSLI at Stanford University a most enlightening and comfortable experience. I also want to thank Yi Zhang for sharing both his robust meaning composition framework and parsing results for the Jigsaw parser.

Further, I would like to extend my thanks to GSLT (Swedish National Graduate School of Language Technology) for facilitating and sharing their excellent courses, which have been most valuable for me during this PhD. Experimentation and engineering was made possible through access to the TITAN high-performance computing facilities at the University of Oslo (UiO), and I am grateful to the Scientific Computation staff at UiO, as well as to the Norwegian Metacenter for Computational Science. This project relies heavily on a range of open source software, and I am most grateful for the developers who make their software and source code freely available.

Finally, I would like to thank Mareike for constantly reminding me that, regardless of the importance of deep semantic grammars, there are after all more important things in life.

Contents

Chapter 1

Introduction

This PhD thesis is devoted to one of the fundamental aspects within *Natural Language Processing* (NLP), namely the task of automatically annotating natural language utterances with grammatical analyses – *parsing*. Specifically, the thesis investigates so-called *deterministic* parsing approaches scaled for broad-coverage, semantically rich, unification-based precision grammars.

Deterministic and near-deterministic parsing is fundamentally different from traditional, exhaustive approaches that aim to retrieve a set of alternative grammatical analyses for each utterance. The common paradigm for exhaustive and near-exhaustive parsing approaches, *chart parsing*, applies some type of dynamic programming algorithm to derive the most probable analysis for an input sentence through the generation of a chart that holds a set of candidate analyses. For most use-cases, only one single analysis will be utilized, namely the analysis which has the highest probability of being considered 'correct', while the alternative analyses that are retrieved, and which are also licensed by the grammar, will be disregarded.

Deterministic parsing can mitigate both the time and space complexity challenges often associated with chart parsing methods. The costly construction of a parse forest is circumvented, because a deterministic parsing approach will be devoted to deriving one single analysis, typically as it reads the input sentence one word/token at a time. This is typically achieved through a greedy parsing algorithm that applies a classifier in order to execute a sequence of locally optimal choices, which we will refer to as *transitions*. This sequence of transitions defines the syntacto-semantic derivation of the utterance, which upon completion approximates the globally optimal solution.

There are several attractive features to this approach. The time-complexity will typically be linear when the algorithm is deterministic, i.e. when it does not allow for later changes to the partial derivation. For a number of applications, e.g. speech recognition, the ability to process input and provide

partial analyses on the fly per word, and not per sentence, can also be vital.

The drawback, or rather, the biggest challenge associated with this parsing strategy is that the classifier has no fault tolerance – if the parser commits to an incorrect parsing decision, this would imply that the correct analysis will not be retrieved.

1.1 Research Questions

The efficacy of a parsing technique is tightly coupled with factors such as the language and grammatical framework which are targeted. Given that the success rate of a deterministic parsing approach directly relies on its ability to predict and execute a sequence of 'correct' transitions, the performance of the parser is essentially determined by its ability to correctly navigate though the search-space of parsing decisions.

Transition-based deterministic parsers have proven themselves competitive with traditional chart parsing approaches for grammatical frameworks such as *Dependency Grammar* (Tesnière, 1959), *Combinatory Categorial Grammar* (CCG; Steedman, 2000) and *context-free grammars* (CFG; Chomsky, 1956). Parsing for semantically richer grammars would however entail a very different search-space since these grammars make more fine-grained semantic distinctions regarding how words and phrases relate form to meaning (this is further discussed in Oepen et al., 2002a, p. xvi), something which could potentially (but not necessarily) pose greater challenges for a deterministic parsing approach, compared to traditional chart parsing methods. This is the area in which we will focus our investigation in this PhD project, and we can hereby define four research questions to which we seek an answer in this thesis:

1. Can transition-based, deterministic parsing be scaled up to a broad-coverage precision grammar like the English Resource Grammar (ERG; Flickinger, 2000)?
2. Which type of linguistic resources are required in order to enable a data-driven deterministic parsing system for this kind of grammatical formalism, and to which scale?
3. To what degree do techniques and experiences applied in other frameworks transfer to this setup?
4. What trade-offs (e.g. between parser precision, robustness, and efficiency) can be observed?

In this PhD thesis we will develop a parsing platform which addresses these currently unresolved research questions. Through the development of this platform, dubbed *CuteForce*, we aim to investigate the feasibility of applying a transition-based, deterministic parsing approach to a large-scale, handcrafted, unification-based precision grammar. Specifically, we have targeted

ERG, which belongs to a multi-lingual family of DELPH-IN (Deep Linguistic Processing with HPSG Initiative)[1] grammars, which are hand-crafted, semantically deep, head-driven phrase structure grammars.

1.2 Motivation

While statistical parsing for CFG has been targeted by research initiatives for more than 40 years (Probabilistic Context-Free grammar was introduced by Booth, 1969), parsing with larger unification-based grammars has a more recent history (Kaplan and Maxwell III, 1996; Torisawa and Tsujii, 1996; Uszkoreit et al., 1994; Gawron et al., 1982, inter alios). Unification-based grammars constitute an increased complexity compared to e.g. CFGs, and parsing with large-scale unification-based grammars may at times have been considered prohibitively expensive (this is further discussed in Oepen et al., 2002a, p. xx, and Maxwell III and Kaplan, 1996). Through advances in hardware and innovative parsing algorithms, this has however to a large extent changed, and with the aforementioned research initiatives multiple systems for parsing with broad-coverage unification-based grammars have emerged since the mid 90s.

Despite these advances in hardware, parsing algorithms and methods for distributed computing, there will however be applications and use-cases for which a traditional chart parsing scheme may be impractical or undesirable, regardless of the grammatical framework in question. Possibly as a response to this, we have over the last decade seen an increased interest in deterministic and near-deterministic transition-based parsing approaches that circumvent the costly construction of a parse forest, having the potential to be substantially more efficient than traditional chart parsing approaches.

There are, as we see it, at least three strong motivations for advancing the research in (near-)deterministic, transition-based parsing:

1. **Efficiency:** A deterministic parsing algorithm reduces parsing to a sequence of classification choices (or transitions) where the length of this sequence is linearly bounded by the input length. Even if moderate non-determinism is allowed (e.g. through backtracking), near-deterministic parsing has the potential to be far more efficient than chart parsing approaches that opt to construct the full parse forest.
2. **On-the-fly processing:** A (near-)deterministic parsing algorithm builds a partial derivation during processing. In itself, there may be use-cases where a partial derivation is required, as opposed to delaying the processing until a sentence boundary is reached. For certain applications, such as speech processing, this would in principle mean

[1] `http://www.delph-in.net/`

that we could utilize real time processing, rather than delaying the processing until the end of the sentence.

3. **Feature functions:** As a (near-)deterministic transition-based parser builds partial derivations during processing, it facilitates the use of globally, as well as locally defined *feature functions*. Globally defined feature functions can increase the precision of the parser as they provide a layer of granularity that not easily available in a packed parse forest, due to the restriction of locality: "This is because parsers segment parse trees into constituents and factor equivalent constituents into a single constituent (edge) in a chart." (Ninomiya et al., 2011, p. 332)

As we mentioned earlier, deterministic parsing approaches to CFG, dependency grammar, CCG, inter alia, have obtained encouraging results both in terms of parsing accuracy and efficiency. Given the efficiency challenges associated with constructing the full parse forest for very large unification-based grammars, this provides, as we see it, an additional incentive for advancing the research on deterministic parsing for such grammars. Even if state-of-the-art accuracy is not obtained, deterministic and near-deterministic parsing approaches to large-scale unification-based precision grammars could potentially be an enabler that would license the use of deep semantic parsing in a range of areas where traditional parsing approaches have proven to be too inefficient.

1.3 Contributions

In order to develop a capable data-driven deterministic parsing platform targeting large-scale precision grammars like the ERG, we require a sufficient amount of training data, and a preprocessing stage that produces the input buffer for the transition-based parser. We can see this project as four separate sub-tasks, where each element relies on the preceding segment(s). Hence, we will here give a brief introduction to the four main components of this project:

Augmenting Linguistic Resources

Although approximately 30,000 sentences of annotated data for the ERG were available in the so-called Redwoods Treebank (Oepen et al., 2002b) at the beginning of this project, it soon became clear that a data-driven parsing system of our aspirations would benefit from an amount of training data of a much larger scale. In association with the *WeScience Initiative*,[2]

[2]The WeScience Initiative is a collaboration between the University of Oslo, the Center for the Study of Language and Information at Stanford University, the German Research Center for AI, and Saarland University: `http://moin.delph-in.net/WeScience`.

the first part of this project was therefore devoted to the extraction and development of new annotated corpora derived from Wikipedia, namely the *WeScience Treebank* and *WikiWoods* (Ytrestøl et al., 2009; Flickinger et al., 2010, see Chapter 3). These new linguistic resources address the fastest growing segment of the Web, namely user-generated content, and we expect that they can enable further research within this domain.

Supertagger

Like many existing parsing systems, our parser requires a preprocessing stage that equips the tokens in the input utterance with fine-grained lexical categories, and we refer to these preterminals as *supertags*. Supertagging has been applied for a range of applications, either to increase the precision of a parser, or by reducing the search space, using supertagging as an initial disambiguation stage.

We evaluated a number of alternative configurations, both with respect to machine learning algorithms and the use of training data. The newly derived linguistic resources from Wikipedia allowed us to perform an in-depth study of how alternative learning algorithms can utilize variable amounts of gold standard and 'silver standard' (i.e. automatically annotated) training data – a research question which had not been adequately addressed within the ERG due to lack of linguistic resources prior to our release of WikiWoods.

Deterministic HPSG Parsing

Taking advantage of the input buffer generated by the preprocessing and supertagging stage, we have developed a transition-based deterministic parsing platform targeting large-scale, semantically deep HPSGs, and optimized for the ERG. This parsing platform, *CuteForce*, is to our knowledge the first attempt to adapt a deterministic transition-based parsing algorithm to a DELPH-IN (introduced in Chapter 3) grammar. (Ytrestøl, 2011a)

In a transition-based parsing scheme, the syntacto-semantic analysis of the input utterance is expanded through a sequence of deductive parser actions, i.e. transitions, where a classifier selects each transition that is executed. This classifier is trained using pre-existing annotated corpora, and linguistic resources made available though the release of the WeScience Treebank and WikiWoods, and we survey experimentally how the use of training data from alternative domains and in varying amounts impact the accuracy of the parser.

Near-Deterministic HPSG Parsing

In addition to a deterministic parsing algorithm, we have developed and evaluated a near-deterministic overlay that can be used to enable moderate

backtracking for a transition-based parsing system (Ytrestøl, 2011b). This overlay is adapted to CuteForce, allowing us to evaluate the parsing platform both in deterministic mode, and with moderate backtracking, used in order to repair an incorrect parsing decision and thereby directing the parser back to the correct search path. A novelty in our backtracking method is the use of *ranking* when selecting the backtracking point, enabling the backtracker to make use of syntacto-semantic information from the partially built analysis, which is a layer of information that is inaccessible for traditional backtracking approaches.

1.4 Thesis Outline

The structure of this thesis reflects the pipeline of dependencies that the components of the thesis have on one another.

Chapter 2: Background

In this chapter we give a birds-eye view of parsing, emphasizing the conceptual differences between chart parsing and deterministic processing. Further, we give an introduction to HPSG, and parsing within the HPSG framework. Finally, we survey other initiatives that relate to ours, either from a methodological point of view, or based on the grammatical framework on with which we work.

Chapter 3: ERG and DELPH-IN Resources

In this chapter, we introduce the LinGO English Resource Grammar (ERG), which will be the HPSG to which we adapt our parsing platform. Further, we present the corpora and treebanks which we have developed and/or used in this project. All the data resources conform to the ERG, and are maintained under the DELPH-IN umbrella.

Chapter 4: Supertagging

This chapter presents the preprocessing that comes prior to the parsing stage. The aim of the preprocessing stage is to provide the CuteForce parser with a tokenized input buffer consisting of a supertag and a part-of-speech tag for each token. In this chapter the development of this preprocessing stage is described, and the accuracy of the supertagger is evaluated in isolation.

Chapter 5: Deterministic HPSG Parsing

Chapter 5 presents our deterministic transition-based parsing platform for HPSG, dubbed CuteForce. This system is both generic and parameteriz-

able, optimized for the ERG, and assumes an input buffer provided by the supertagger. We present the parsing algorithm, and introduce the machine learning model, i.e. the classifier that is used to guide the transition-based parser. We have investigated several alternative configurations for deterministic transition-based parsing with DELPH-IN grammars, and these will be presented in this chapter.

Chapter 6: Near-Deterministic HPSG Parsing

In this chapter we present a near-deterministic overlay to a deterministic transition-based parsing approach. This overlay, dubbed *Optimistic Backtracking*, is heavily constrained to minimize computational complexity, and we describe how this approach can be used to enable moderate backtracking for a deterministic parsing system. We present and discuss two different strategies for determining the point for backtracking. Most prominently, we present an innovative backtracking strategy that applies ranking when selecting backtracking points. This method is compared to a more conventional backtracking strategy, and the accuracy of both methods is evaluated and discussed.

Chapter 7: Results and Evaluation

This chapter begins by addressing parser evaluation in general. Further, we present and evaluate the results from our own parsing system, discussing the results from the deterministic and near-deterministic configurations. By comparing the overall parsing performance and efficiency of CuteForce with parsers using similar frameworks, we seek to determine how our parsing platform relates to existing parsing systems.

Chapter 8: Summary and Conclusion

In this chapter we provide a condensed overview of the insights and results our project has produced, and which are captured in this thesis. Further, we will address aspects that we think may benefit from additional investigation, but which have fallen outside of the scope of this project, and outline some ideas on how further refinement could increase both performance and efficiency for the parsing platform we have described.

Chapter 2

Background

In this chapter we will give a brief introduction to the fundamental aspects of parsing. Although we assume that the reader will have a clear grasp of the cornerstones of our field, we find it useful to give this introduction both for completion and in order to establish a common set of terms.

We focus primarily on chart parsing, which has been the dominating paradigm over the last decades, together with determinstic- and near-deterministic transition-based parsing, which pertains to this project specifically. In terms of linguistic frameworks, we give a short account of *Context-Free Grammar* (CFG), which is the most widely used mathematical system for modeling constituent structures. Then we shift the focus to frameworks that are of particular significance to this project. This includes *Head-Driven Phrase Structure Grammar* (HPSG) in particular, which will be used by our parsing platform. We will also give a brief introduction to *Dependency Parsing*, which has been an important framework within research on deterministic parsing.

We will particularly stress the division between exhaustive and near-exhaustive chart-based parsing approaches and deterministic and near-deterministic transition-based algorithms, where we will contrast state-of-the-art chart parsers to (near-)deterministic approaches for a range of linguistic frameworks. Special attention will be given to efforts within HPSG.

2.1 Natural Language Processing

The aspiration of having a computer make sense of human language was sparked at the very beginning of the computer science era. Indeed, machine translation was one of the very first initiatives in the field that was later to be known as *Natural Language Processing* (NLP).

The term *Natural Language Understanding* was coined in the 1960s targeting efforts to make computers respond to human language input – a task which minimally required an understanding of syntax and syntactic rules,

together with a dictionary. An example of this is the SHRDLU program (Winograd, 1972). As language processing tasks grew in complexity, the need for a framework to represent syntax became apparent. Particularly central in the pioneer era was the development of *Context-Free Grammar* (Chomsky, 1956), advanced in the mid-50s by Noam Chomsky.

2.2 Context-Free Grammar

Context-Free Grammar (CFG) is the most commonly used mathematical system for modeling of constituent structure. For completion we will give a short account of the fundamental aspects of the formalism.

A CFG is a set of rules describing how words of a language can be clustered to phrases. Formally, it may be represented as a four-tuple $< N, \Sigma, R, S >$ where N is the set of *non-terminal* symbols, Σ is the set of *terminals* in the language, R is the set of *rules*, of the form $A \rightarrow$ ß, where A is a non-terminal and ß is a string of symbols from the set $(N \cup \Sigma)$*, and S is the start symbol.

Table 2.1 provides an example of a simple context-free grammar, and Figure 2.1 presents a corresponding derivation tree for the sentence (1).

(1) the girl sleeps

The non-terminals and terminals in the grammar are printed in uppercase and lowercase respectively.

S	→	NP VP
NP	→	DET N
VP	→	V
DET	→	the
N	→	girl
V	→	sleeps

Table 2.1: Simple CFG grammar and lexicon.

2.2.1 Probabilistic Context-Free Grammar

A common augmentation to a CFG is PCFG, *Probabilistic Context-Free Grammar* (also known as *Stochastic Context-Free Grammar*, Booth, 1969). For each rule in R, a conditional probability $P(A \rightarrow$ ß$)$ is assigned. To the grammar in Table 2.1 we can add a transitive verb phrase rule, hence introducing ambiguity to the grammar (see Table 2.2). For a sentence S and an analysis T, the probability for the analysis can be formalized:

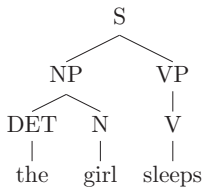

Figure 2.1: Example of a simple phrase tree for *the girl sleeps*, given a small lexicon and a set of rules (Table 2.1).

VP	→	V	(0.4)
VP	→	V NP	(0.6)

Table 2.2: Transitive and intransitive VP rules with corresponding conditional probabilities.

$$P(T, S) = \prod_{R \; in \; T} P(R) \tag{2.1}$$

where the analysis T consists of rule instantiations R which carry individual conditional probabilities. The most efficient way of deriving the optimal analysis is typically achieved through dynamic programming.

2.2.2 Dynamic Programming

For parsing with a (P)CFG, a number of dynamic programming algorithms have been proposed, most prominently CKY (Cocke, 1969; Kasami, 1965; Younger, 1967), Earley algorithm (Earley, 1970) and Chart Parsing (Kay, 1980). Essential to dynamic programming is the storing of partial analyses in a table, thus we are able to reuse these partial analyses to form larger constituent structures without recomputing previous analyses.

Table- or chart parsing based approaches have dominated probabilistic parsing over the last decades. This will be further discussed in Section 2.5.

2.3 Dependency Grammar

Although CFG traditionally has been the most influential grammatical framework for annotating human languages, *Dependency Grammar* has roots back to ancient Greek and Indian, and the modern framework is heavily founded on work by Lucien Tesnière (Tesnière, 1959). Dependency parsing has been applied in a variety of applications, amongst others relation extraction (Culotta and Sorensen, 2004), information retrieval (McClosky et al., 2011) and

machine translations (Ding and Palmer, 2005). Being comparatively easy to train and parse,[1] and providing predicative-argument relations, which for most NLP applications are the most significant features, dependency parsing has become increasingly popular within the NLP community recently.

Rather than phrase-structure rules, dependency grammar encodes the relations of head words and their dependents:

> Since a dependency representation consists of lexical elements linked by binary asymmetrical relations, it can be defined as a *labeled directed graph*, where the set of nodes (or vertices) is the set of lexical elements (as defined by the particular framework), and the set of labeled arcs represent dependency relations from heads to dependents. (Nivre, 2005, p. 8)

The unlabeled dependency relation for the sentence *sandy kicked the ball* is represented in Figure 2.2. In this sentence, *kicked* takes *sandy* and *ball* as its dependents. We also see the head-dependent relation between *ball* (head) and *the* (dependent).

Figure 2.2: Unlabeled dependency relations for the sentence *Sandy kicked the ball.*

For languages of freer word order, dependency parsing may prove more flexible than CFG, since it captures the notion of relations between words regardless of the word order in the sentence.

Typed Dependency A common augmentation to dependency grammar is to label the dependencies from a fixed inventory of grammatical labels. Figure 2.3 provides relational labeling between the head and dependents of the sentence from Figure 2.2.

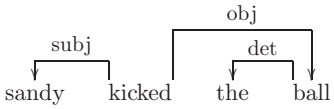

Figure 2.3: Labeled dependency relations for the sentence *sandy kicked the ball.*

[1] This is more elaborately discussed in Covington (2001, p. 3).

In the sentence in Figure 2.3, the example sentence is equipped with labeled head-dependent relations, annotating the subject and object of the head word, and the determiner relation between the dependent *the* and the head *ball*.

2.4 Head-Driven Phrase Structure Grammar

> A sign is the basic unit of language (a given language at a given time). Every language is a complete system of signs. Parole (the speech of an individual) is an external manifestation of language. (Ferdinand de Saussure)

Head-driven Phrase Structure Grammar (HPSG; Pollard and Sag, 1994) as a grammatical framework can be seen as an extension of *Generalized Phrase Structure Grammar* (GPSG; Gazdar et al., 1985), but is also influenced by a number of other formalisms, like *Lexical Functional Grammar* (LFG; Bresnan and Kaplan, 1982). Together with LFG, HPSG shares the principle idea that a large part of the grammar in a language is stored in the lexicon and the lexical entries. The HPSG notion of *sign* in Ferdinand de Saussure's sense is derived from *semiotics*. Although Saussure did not invent the term *semiotics*, he is seen as one of founding fathers within the field.

2.4.1 Lexical and Phrasal Signs

An HPSG grammar consists of two instances of *signs* – lexical and phrasal signs. These signs are normally represented as *Attribute-Value Matrices* (AVMs), typically referred to as a *Typed Feature Structure* (see Carpenter [1992] for a comprehensive introduction to Typed Feature Structures):

1. Lexical signs, containing information on the representation of the word (either pronunciation *PHON* or spelling *ORTH*) and the syntactic and semantic properties of the word (*SYNSEM*).
2. Phrasal signs, representing the syntactic and semantic properties of the phrase, and similar to a constituent in CFG.

Parallel to LFG, HPSG derives most of the syntactic constraints from the lexical signs. Historically HPSG consisted of a very limited number of phrasal rule schemata, although recent grammars in HPSG frequently exhibit a richer inventory of rules.

2.4.2 Inherent Properties and Constraints

An HPSG derivation is typically expressed through a typed feature structure. In the following, we will present a few fundamental grammatical aspects that are regulated through the inherent constraints of HPSG. For a more comprehensive overview, we refer to Pollard and Sag (1994).

Valence Features *Valence Features* (VAL) are attributes in the feature structure that encode the clauses with which a word can or must combine. These syntactic possibilities or constraints are encoded as feature structure attributes to VAL. COMPS (complements) have a defined list of attributes used to encode the combinatoric potential of the word. For simplicity, we can use *itr* (intransitive) and *str* (strict transitive) for a verb relation. Consider the (simplified) HPSG sign for *sleeps* (see Figure 2.4).

$$\begin{bmatrix} \text{sleeps} & \\ \text{HEAD} & \text{verb} \\ \text{VAL} & \begin{bmatrix} \text{val-cat} \\ \text{COMPS itr} \end{bmatrix} \end{bmatrix}$$

Figure 2.4: HPSG sign for the intransitive verb *sleeps*

While COMPS is a generalization for the attachment of objects, HPSG also uses a generalization for determiners and subjects, namely SPR (specifier). The boolean value of SPR encodes whether a specificer is required or not, specifically with the attribute value pairs $[SPR \quad -]$ and $[SPR \quad +]$, respectively. We can expand the feature structure for *sleeps* in Figure 2.4, conveying the intuition that the verb requires a specifier (in this case a subject), as in Figure 2.5.

$$\begin{bmatrix} \text{sleeps} & & \\ \text{HEAD} & \text{verb} & \\ \textbf{VAL} & \begin{bmatrix} \text{val-cat} & \\ \text{COMPS} & \text{itr} \\ \text{SPR} & \text{-} \end{bmatrix} \end{bmatrix}$$

Figure 2.5: HPSG sign with the valency constraints (**VAL**) for the verb *sleeps*.

The Valence Principle in HPSG states that the VAL features are identical for the mother and the head daughter, unless the phrasal rules state otherwise. This brings us to another cornerstone of HPSG, namely the *Head Feature*, which will be addressed below.

The Head Feature Principle Key to HPSG is the *Head Feature Principle*, stating that in a *headed phrase*, one single daughter is designated as the *head daughter*. In these headed phrases, the mother and the head daughter share the same HEAD value in their feature structures. Typically, for the

headed phrase VP, the head daughter will be the verb, for an S phrase the head daughter is the VP, etc.

The Agreement Feature Agreement relations between words and phrases in a sentence are defined through a feature AGR (agreement). For English this includes PER (person) and NUM (number), for other languages this list could be expanded, for instance with case marking. The agreement constraints are passed up from words to phrases following the head feature principle.

$$\begin{bmatrix} \text{sleeps} & & \\ \text{HEAD} & \begin{bmatrix} \text{verb} & \\ \textbf{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PER} & \text{3rd} \end{bmatrix} \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{val-cat} & \\ \text{COMPS} & \text{itr} \\ \text{SPR} & \text{-} \end{bmatrix} \end{bmatrix}$$

Figure 2.6: HPSG sign with the agreement constraints (**AGR**) for the verb *sleeps*, ensuring that the subject will be 3rd person singular.

In Figure 2.6 the agreement constraints for *sleeps* are encoded in the feature structure, and ensure that the subject will be 3rd person singular.

2.4.3 Unification

HPSG is, together with LFG, the most prominent unification-based grammatical framework. Similar to CFG, HPSG rules determine how the lexical items in the sentence can be combined with other items to form constituents described by the grammar. While these production rules have a strong resemblance to CFG rules (see Table 2.1), any item and grammatical rule is augmented with a typed feature structure. The application of a rule is then augmented with a constraint check, i.e. a test as to whether the application of the rule leads to a successful unification of the applicable feature structures. If the unification succeeds, a new typed feature structure is derived.

$$\begin{bmatrix} \text{ARG} & \begin{bmatrix} \text{PER} & \text{1st} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{ARG} & \begin{bmatrix} \text{NUM} & \text{sg} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{ARG} & \begin{bmatrix} \text{PER} & \text{1st} \\ \text{NUM} & \text{sg} \end{bmatrix} \end{bmatrix}$$

Figure 2.7: Successful Unification

A typed feature structure can be merged with another consistent feature structure if they do not have conflicting attribute-values. Hence, the attribute-values impose the constraints of the grammar rule. In Figure 2.7 we see the successful unification of two feature structures, where a new merged feature structure is formed.

$$\begin{bmatrix}\text{ARG} & \begin{bmatrix}\text{PER} & \text{1st}\end{bmatrix}\end{bmatrix} \sqcup \begin{bmatrix}\text{ARG} & \begin{bmatrix}\text{NUM} & \text{sg} \\ \text{PER} & \text{3rd}\end{bmatrix}\end{bmatrix} = \text{FAIL}$$

Figure 2.8: Unsuccessful Unification

In Figure 2.8 unification fails because the attribute value of PER is inconsistent for the two feature structures. Such unification failure could occur when trying to combine the lexical sign for *I* (1st person singular) and *swims* (3rd person singular) in a Head+Subject phrase. Phrasal rules ensure that the word signs that comprise the phrase, and ultimately the sentence, are licensed by the grammar.

2.4.4 Syntactic Analysis in HPSG

Pioneer work within Computational Linguistics focused mainly on parsing Context-Free Languages. The shortcomings of strict CFG for natural languages are however obvious. Consider sentence (2) and (3):

(2) The woman sleeps.

(3) *These woman sleep.

A simple CFG rules like:

(4) NP $\rightarrow$ Det Noun

would not be able to address the agreement constraint that is violated in sentence (3). Writing specific rules could be a fix, for instance by introducing a rule like:

(5) $\text{NP}_{plur} \rightarrow \text{Det}_{plur}\ \text{Noun}_{plur}$

However, multiplying each such rule would cause a tremendous inflation to the grammar. It would also fail to capture the linguistic principle behind agreement, namely that the agreement values of the daughters should not be contradictory. Unification-based grammars take care of this issue by encoding the constraints directly in the grammatical items and lexical signs.

Given the sentence *Sandy sleeps*, we can apply the Head+Subject phrase rule and the NP rule derived from the grammar to reach the full HPSG analysis for the sentence (Figure 2.9).

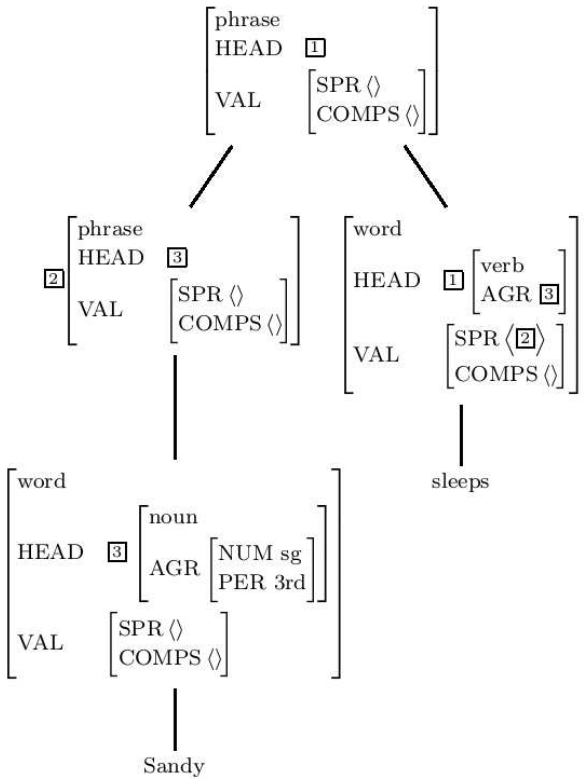

Figure 2.9: Successful application of the Head+Subject rule yields a valid parse for the sentence *Sandy sleeps* (simplified analysis).

Figure 2.9 shows a (simplified) well-formed HPSG structure for the sentence *Sandy sleeps*. A corresponding CFG structure, reflecting the unary and binary rules used by the HPSG grammar, is presented in Figure 2.10. Note that each projection in the derivation undergoes an implicit constraint check, ensuring that the attribute-values of the concerning feature structures are unifiable. Figure 2.11 presents the English Resource Grammar (ERG) analysis for the same sentence (ERG will be introduced in Chapter 3).

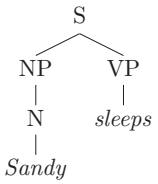

Figure 2.10: CFG tree projecting the HPSG analysis in Figure 2.9

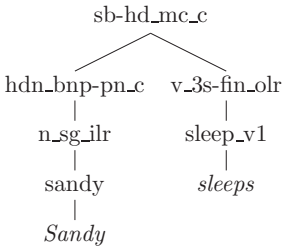

Figure 2.11: English Resource Grammar analysis of the sentence *Sandy sleeps* from Figure 2.9

For heavily lexicalized grammars like HPSG, where the lexical feature structures carry a large amount of information, computational processing can be very expensive. To avoid reentering previously traversed search paths, the parser will need to keep track of which parts of the search space it has already computed. By employing a table for storing previously parsed subtrees, the parser would be able to do table lookup before parsing a given subtree in the sentence, avoiding the computational cost of reparsing the segment. In Section 2.5 we will outline the most popular technique for storing and utilizing previously parsed subtrees, namely *chart parsing*.

2.5 Chart Parsing

Chart parsing is a parsing approach advanced by Martin Kay (Kay, 1980) and Ronald M. Kaplan (Kaplan, 1973). Similar to CKY and the Earley algorithm, the chart parsing algorithm stores parsed items, subtrees, in a table where they can be retrieved for later use. Chart parsing provides an algorithm that permits a flexible determination of when a parse item should be processed, accomplished through the use of an *agenda*. In both CKY and Earley algorithm the ordering in which items are processed is statically determined by the algorithm. However, in chart parsing the agenda allows for a flexible work queue that can be determined as a separate component to the parser. The items, or partial analyses of the derivation are called *edges*, derived from graph theory with the intuition that each input token represents a vertex in a graph. The *fundamental rule* is the key principle for edge processing in chart parsing. Briefly, it states that if we have two contiguous partial analyses, edges, "where one of the edges provides the constituent that the other one needs, a new edge should be created that spans the original edges and incorporates the provided material." (Jurafsky and Martin, 2008, p. 449) If the agenda is empty, parsing stops, and the chart is returned. If one or more edges span over the entire input sentence, and has a valid root label, the parser has found a valid analysis.

By introducing the agenda as an independent component to the parser, chart parsing facilitates extensive adaptability with respect to parsing with various grammatical frameworks. Whereas the backbone of the agenda, and the actual constraint-checking inherent in any parsing system will be different for the individual grammatical frameworks and parsing algorithms, the storing of edges in the chart, and the application of the fundamental rule has been applied with a large number of grammatical frameworks.

The term *chart parsing* is now widely used for any parsing approach that stores subanalyses in a table, which can later be retrieved and utilized, e.g in a dynamic programming algorithm. We will assume this wide definition of chart parsing for the remainder of this thesis.

The Chart

Fundamental to any CFG chart parser is the ability to store partial analyses, subtrees. We will give a brief presentation of a canonical dynamic parsing algorithm, namely CKY parsing for CFG. The CKY algorithm assumes a grammar in Chomsky normal form (CNF), requiring that all rules of the grammar G must be a non-terminal projection $A \rightarrow BC$ or a terminal projection $A \rightarrow w$, where w is a word.

In Tables 2.3 and 2.4 a simple grammar and a corresponding chart for the sentence *Book that flight* are outlined. The cells in the table are indexed according to the span of the sentence they cover, and the chart will generally

S	→	VP NP
S	→	NP VP
VP	→	V
VP	→	V NP
NP	→	DET N
V	→	book
N	→	book
Det	→	that
N	→	flight

Table 2.3: Simple CFG grammar.

VP,V,N 0,1	 0,2	S,VP 0,3
	DET 1,2	NP 1,3
		N 2,3
'book', 0,1	'that',1,2	'flight',2,3

Table 2.4: A CKY chart for the sentence *Book that flight* using the grammar in Table 2.3.

have $\frac{n(n+1)}{2}$ chart cells, where n corresponds to the number of words in the sentence. In a bottom-up parse, each column in the chart is filled from the bottom, starting with the terminal. For the word *book*, we see that according to the grammar, the terminal can be expanded to an `NP`, `N` and `V` (cell {0,1}). As there are no edges on the left-hand side, we move to the next column. After filling in the terminal projection in cell {1,2}, we move up to cell {0,2}. However, the grammar does not license a new edge spanning {0,2}, because none of the constituents in cell {0,1} can be extended to a new constituent by the incorporation of the constituent in cell {1,2}.

In the third column, we apply the `NP` → `DET N` rule in cell 1,3 by combining `DET` in cell {1,2} with `N` in cell {2,3}. Similarly we derive the `S` and `VP` in cell {0,3} by combining the non-terminal rules in cell {0,1} with the `NP` in cell {2,3}.

If we successfully derive the `S` symbol spanning over all items in the input, we have found a valid parse for the sentence, hence proving that the input is grammatical. The time complexity for such an acceptor is $O(n^3)$, where n equals the length of the input string. However, an exhaustive extraction of the full parse forest has a theoretical worst-case run-time complexity of $O(2^n)$ (exponential).

2.5.1 Chart Parsing for (P)CFG

Many of the state-of-the-art CFG parsers today use a chart parsing scheme. Indeed, the chart parsing algorithm was originally outlined for context-free grammars. Most of the English CFG parsers today take advantage of one of the major phrase structure treebanks available (most prominently Penn Treebank, Marcus et al., [1993], see also Section 3.1 for a more in-depth discussion of treebanks). The treebanks are applied to derive a probabilistic language model.

Lexicalization In Section 2.2 we gave a brief account of the CFG framework. As mentioned in Section 2.2 there is at least one potential shortcoming with a naive CFG formalism: It disregards lexical information when determining the probabilities of the phrasal attachments. Hence, the probabilities of e.g. a transitive versus an intransitive VP would not be influenced by the lexical form of the verb in the sentence, which could have provided information on the attachment constraints, similar to the COMPS attribute in an HPSG features structure, as we discussed in Section 2.4. A common augmentation to address this shortcoming is *lexicalized CFG*. A lexicalized CFG will, in a similar fashion to the Head Feature Principle in HPSG, percolate the lexical head of the constituent through the tree structure, allowing a parser to generate a probabilistic parsing model where the lexical head is taken into account. In Figure 2.12 this is exemplified by a tree structure with and without lexicalized constituent labels.

PCFG-Approximation for HPSG

Zhang and Krieger (2011) present a corpus-driven approach to inducing a grammar-approximation of an HPSG, namely the English Resource Grammar (ERG). Briefly, by inducing a unlexicalized PCFG, they can parse HPSG using a traditional (P)CFG chart parsing scheme. Their best scoring grammar approximation reached a PARSEVAL (Black et al., 1991) F1 accuracy of 84.13%, and they conclude that:

> The high robustness of the PCFG suggests it is a viable way of achieving full coverage parsing with the hand-written deep linguistic grammars. (Zhang and Krieger, 2011, p. 198)

The parser, dubbed *Jigsaw*, is of particular interest to us because it follows some of the intuitions that are fundamental for our own project. By extracting a conventional PCFG from annotated corpora, they apply a parsing approach which has predominantly been associated with more shallow grammatical frameworks, especially CFG. Similar to our parsing approach, presented in Chapter 5, they disregard the actual grammar, ERG, aiming to learn grammatical patterns through the corpora that conform to

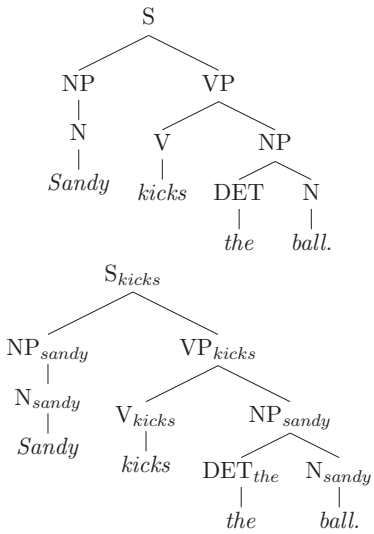

Figure 2.12: Lexicalized and standard CFG tree for the sentence *Sandy kicks the ball.*

the ERG. The parsing accuracy/efficency of Jigsaw will be evaluated and compared to our own system in Section 7.7.2.

2.5.2 Chart Parsing for Unification-Based Grammars

Dynamic programming algorithms like chart parsing used in CFG parsing cannot be applied straightforwardly to unification-based grammars like HPSG. To exemplify: For a PCFG a VP in the chart may consist of multiple individual analyses, but it could still be represented uniformly in the chart with a given probability (namely the highest probability from the candidate analyses). In HPSG there would however be no simple way of collapsing alternative analyses in a chart, because the predicate-argument structures in unification-based grammars involve non-local dependencies defined in the feature structures.

Because every deductive parsing step involves a unification, an exhaustive extraction of the entire parse forest, i.e. all possible parse derivations licensed by the grammar, would result in a very large number of unifications for a medium-length sentence. Since unification is a destructive operation, a non-deterministic parsing algorithm will need to preserve the original feature structure in order to maintain the entire parse forest intact.

A large number of research initiatives have been carried out to address-

ing these issues, as this could potentially be a major bottleneck for non-deterministic HPSG parsing. Miyao and Tsujii (2008) propose a *feature forest model*, "a packed representation of a set of an exponential number of tree structures." This allows for probabilistic modeling without unpacking the feature forest, and a dynamic programming algorithm is proposed. The space-complexity problem can be mitigated by collapsing equivalent parts of the features structures, hence avoiding redundancy by storing a large number of similar feature structures in the parse forest (Zhang et al., 2007; Carroll and Oepen, 2005; Oepen and Carroll, 2000; Callmeier, 2000; Miyao, 1999). Efficient packing of feature structures will not only greatly improve the space efficiency, but typically also increase the speed of the parser. Additionally, doing a full unification for each deductive step may not be necessary, as the verification of the unification constraints may be done only to newly introduced items in the feature structures.

While the theoretical time-complexity of an exhaustive extraction of the full parse forest of any grammar is exponential, a number of techniques have been proposed and applied to avoid pursuing partial analyses that will not, further down the line, lead to a well-formed analysis, hence pruning large parts of the search space. It is fruitful to distinguish between algorithms that aim to (1) exhaustively derive the entire parse forest, and (2) algorithms that do some pruning in order to speed up the parsing, where it cannot be guaranteed (or desirable) that all analyses licensed by the grammar will be retrieved. This will be elaborated in the next section.

Chart Pruning

Exhaustive construction of the full parse forest was long considered an infeasible parsing strategy for unification-based parsing. However, through advances in hardware and algorithms, this is no longer the case. Nevertheless, most high-speed (<500ms per sentence) unification-based parsers do employ some kind of speed-up to avoid the massive complexity problem associated with an exhaustive construction of the parse forest.

Clearly, the most desirable parsing strategy would be to disregard all incorrect/ill-formed analyses, jumping straight to the correct one. Somewhat far-fetched-sounding, this is actually the key principle behind deterministic parsing (see section 2.6). There are however many efforts that fall somewhere in between. In such cases, the idea is to identify ill-formed or incorrect analyses at an early stage, prune them from the parse forest, and hence reduce the inherent complexity.

In a chart parsing scheme, pruning is applied by restricting improbable subtrees from entering the parse chart. This could be done by (1) the use of a supertagger, which would restrict the number of lexical entries that are added to the parsing chart, or by (2) identifying and removing subtrees in the chart parse that are unlikely to contribute to the final result.

2.5.3 PET HPSG Parser

The PET platform was developed as a tool for experimentation with HPSG processing and implementation techniques (Callmeier, 2000). It has further been developed through subsumption-based packing (Oepen and Carroll, 2000), selective unpacking and statistical parse ranking (Zhang et al., 2007). It is maintained by the DELPH-IN consortium, which will be introduced in Chapter 3.

PET employs an agenda-based chart parsing algorithm. For each unification of two feature structures, the chart is checked to make sure the unification is valid. The PET HPSG Parser will, for each input sentence, use a hand-crafted grammar (LinGO ERG, see Chapter 3) to retrieve the candidate analyses that are grounded in the grammar, thus being *well-formed*. During parsing it attempts to create the full packed parse forest for each input sentence, and produces an n-best list based on statistical parse ranking. The PET Parser is today the 'standard' HPSG parser for the DELPH-IN consortium, and its parsing accuracy and efficiency will be compared to the our own system in Section 7.7.1.

System Design Overview

PET is designed to take advantage of the linguistic resources made available by the ERG. In its default algorithm, it can process pre-tokenized input, or use the built-in tokenizer to parse raw text. For each token, a morphological analyzer will derive possible base lemmas for each token. The potential base lemmas of the token are then looked up in the lexicon that maps lemmas to lexical HPSG signs. Each token can typically correspond to multiple lexical signs. All lexical signs grounded in the lexicon are added to the parser chart, and for unknown words underspecified generic entries are added, and later enriched (and thus disambiguated) through contextual information.

After equipping the chart with lexical word signs, agenda-based chart parsing starts according to the rules defined by the grammar. In the default configuration, no probabilistic cutoff is employed, and the parser will extract the full parse forest that is licensed by the grammar. However, a time-constraint is typically set to 60 seconds – if the extraction of the parse forest takes longer than the pre-set time constraint, parsing will time-out (and thus fail) for the given input sentence. Each grammar rule is an implicit unification. Following the feature structure subsumption algorithm developed by Oepen and Carroll (2000), each new edge is checked to see if it subsumes or is subsumed by an existing edge in the chart. This allows for 'packing' of edges, hence circumventing redundant unifications, as it will only check atomic feature structure constraints for two edges once, that way achieving a substantial speed boost and space reduction. There has been a number of research efforts addressing a further optimization of unification-

based parsing specifically aimed at the PET HPSG Parser, i.e. Oepen and Carroll (2002), Carroll and Oepen (2005), inter alios.

The semantic representation is an integrated part of the lexicon. Hence, the parser constructs the semantic derivation along with the syntactic representation. *Minimal Recursion Semantics* (MRS; Copestake et al., 2005) is the de facto semantic representation applied by the DELPH-IN grammars, and can be derived directly from the HPSG analysis provided by PET.

Pruning The PET parser can gain a substantial efficiency increase by reducing the search-space. Dridan (2009) demonstrated that supertagging the input data can contribute to increased efficiency by restricting the number of lexical entries that are added to the parsing chart of the PET parser (supertagging will be further discussed in Chapter 4).

Cramer (2011) introduced a search-space restriction for the PET Parser called *agenda-based task pruning*, evaluating various strategies. The overall principle is to identify and prune parser actions stored in the agenda that are unlikely to contribute to the final result without using a costly unification operation. Cramer (2011, p. 169) reported that "a speedup of 95% can be achieved without negatively affecting f-score." For the PET parser, this strategy could also improve robustness, as it may reduce (or even eliminate) time-outs.

2.5.4 Enju HPSG Parser

Enju (Miyao et al., 2004) is another widely used HPSG for English. Unlike the ERG, it is induced from the Penn Treebank (Marcus et al., 1993). This is achieved through the definition of a few manually written grammar rules that are used to annotated the treebank with "*partially-specified derivation trees.*" (Miyao et al., 2004) Through this approach, they avoid manually defining the individual entries of the lexicon:

> [L]exical entries are automatically extracted from partially-specified derivation trees given as the annotations to the treebank. *Inverse schemata* are applied to each phrasal sign in a partially-specified derivation tree. That is, given a mother as an input to a schema, daughters are computed. (Miyao et al., 2004, p. 689)

According to the authors, this approach also contributes to the robustness of the grammar, because the grammar will be derived from a large-scale treebank that has the potential to capture linguistic corner-cases that could go beyond the intuitions of a grammar developer. We refer to Miyao et al. (2004) for a comprehensive presentation of the acquisition of the Enju grammar.

The Enju HPSG parser is a conventional chart parser in the sense that it extracts the parse forest for each utterance, and disambiguates through the

use of discriminative log-linear models (Miyao and Tsujii, 2005). Further, they developed a *preliminary probabilistic model*, where they prune lexical readings of a word that fall below a probability score cutoff, although being licensed by the grammar. Consequently, the search space is reduced, and the computational cost of deriving the parse forest for the remaining lexical readings decreases. The Enju HPSG Parser is the de facto parser for the Enju grammar, and is intergrated in a number of systems that use the Enju grammar (Miyao and Tsujii, 2008; Matsuzaki and Tsujii, 2008; Hara et al., 2005, inter alios).

2.6 Transition-Based Parsing

While chart parsing is the common paradigm for (near-)exhaustive parsing approaches, deterministic and near-deterministic parsing approaches predominantly employ a *transition-based* parsing algorithm. A transition-based parsing system employs a set of *transitions*, where each transition will typically perform a specific type of manipulation on the data structure of the parser. This will be further elaborated in Chapter 5 (see specifically Section 5.2.3), when we will present a deterministic, transition-based parser for HPSG.

A deterministic parser aims to derive a single analysis given an input string. Since almost any medium to long sentence carries substantial inherent ambiguity, this would in practice mean disambiguating on the fly by making a sequence of local choices that are expected to lead the 'correct' derivation.

Typically this is achieved through some variation of a shift-reduce parser, processing the input string linearly. Deterministic parsing saves a substantial amount of computation as it commits to a single analysis. In search tree theory, this is consistent with a *greedy best-first search* (Russell and Norvig, 2009; Pearl, 1984). Consequently, the algorithm will never spend time on deriving a parse tree or a subtree that will later be rejected.[2]

Feature Functions In addition to its computational efficiency, there is as we see it another strong argument for deterministic and near-deterministic parsing: because a deterministic parsing algorithm maintains and expands a partial syntacto-semantic analysis throughout the processing of the input, this facilitates the use of globally defined *feature functions*. Such feature functions can increase the precision of the parser by providing a layer of

[2]Parsing may however fail, i.e. the parser may be unable to provide a parse. In this case, no parse derivation is provided, and the partial analyses that have been derived during paring will be abandoned. This does however depend on the parser and the framework, and will be further discussed in Section 6.4.4.

granularity that is not easily available in a conventional packed parse forest. Ninomiya et al. (2011) describe this as a *locality* problem:

> Both estimation and parsing with the packed parse forest, however, raise an inherent problem that derives from the restriction of locality. Feature functions can only be defined for local structures, which limit the parsers accuracy. This is because parsers segment parse trees into constituents and factor equivalent constituents into a single constituent (edge) in a chart. (Ninomiya et al., 2011, p. 333)

In Ninomiya et al. (2011) transition-based deterministic parsing is applied to achieve "simple and fast parsing without the mechanisms of the packed parse forest by accurately choosing search paths."(Ninomiya et al., 2011, p. 331)

Inherent Challenges A deterministic transition-based parser's main advantage over exhaustive parsers is that they perform a very limited number of computations. However, for most grammatical frameworks and languages, deterministic parsers still typically obtain a slightly lower accuracy than state-of-the-art non-deterministic parsers. There are a couple of obvious intuitions for this. After all, the natural language as it occurs in newspapers, literature and through speech utterances carry so much ambiguity that even a human would not be able to deterministically disambiguate any given sentence. Garden path sentences, exemplified in sentence (6), are the canonical example of sentences that are typically misinterpreted during reading due to an early incorrect grammatical assumption.

(6) The horse raced past the barn fell.

For sentence (6), if a deterministic parser incorrectly makes the assumption that *raced* constitutes the head verb of the sentence, there would be no recovery when the word *fell* occurs, and it would undermine the syntactic structure the deterministic parser is building. A way to recover from an incorrect grammatical assumption is to allow for non-deterministic search. This will be elaborated in the following section.

2.6.1 Near-Deterministic Parsing

There is a continuum of parsing approaches between a conventional chart parsing scheme that exhaustively considers the full parse forest, and a deterministic configuration that commits to one single analysis. Whereas *chart pruning* (see Section 2.5.2) was mentioned as an approach to shrink the complexity and search space of a chart parser, at the other end of the spectrum

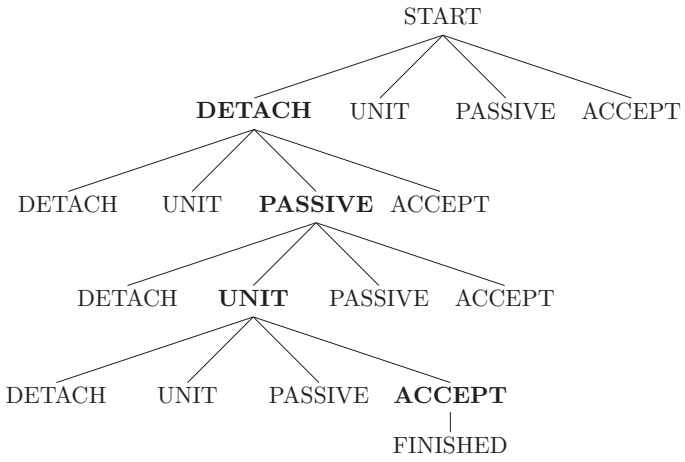

Figure 2.13: Transition-based parsing as a tree-structured search problem, where the transition sequence DETACH, PASSIVE, UNIT, ACCEPT is executed.

there are alternative methods that facilitates near-deterministic transition-based parsing.

Transition-based (near-)deterministic parsing can be described as a tree-structured search problem, to which we can use various methods to explore the search space. We call these methods *Stack Search*, initially advanced by Jelinek (1969). Figure 2.13 represents the tree-structured search space of a transition-based parser with the transitions {`DETACH, UNIT, PASSIVE, ACCEPT`}. When the transitions `DETACH`, `PASSIVE`, `UNIT` and then `ACCEPT` are pursued, the alternative paths are left unexplored. A deterministic configuration prohibits the option of traversing upwards in the tree, exploring an alternative path. If this restriction is lifted, the parsing algorithm is no longer deterministic.

Near-deterministic approaches would often implement some kind of priority queue, where parsing operations could be pushed, and the most probable operation would be popped. These parsing operations will then be nodes in the tree. Indiscriminately adding all parsing operations to the priority queue, and looping through all expansions until the queue is empty would be known as a *uniform-cost search*, guaranteed to retrieve all parse trees. However, this is, due to its inefficiency, an infeasible parsing approach, and non-deterministic parsers would typically employ restrictions to the parsing operations that are added to the priority queue. We will look at three of the most commonly used stack search algorithms in parsing, namely *Beam*

Search, *A* Search* and *Backtracking Search*.

Beam Search Instead of indiscriminately pushing all valid parsing operations to the priority queue, a *Beam Search* defines a *beam width* of parsing operations for each deductive parsing stage, expanding a restricted set of parse extensions to the priority queue. The width of the beam defines how much of the search space that is pruned, and the range can be defined either by a fixed set of expansions for each deductive parsing stage, or by defining a probability range, adding all operations that fall within this range.

A* Search Developed by Hart et al. (1968), *A* Search* is the most widely used *best-first* search algorithm. Similarly to the beam search algorithm, it keeps a priority queue, but this queue consists of the prefix from the root to the current node in the tree, ordered by probability. The algorithm will pop the highest scoring prefix, and evaluate the most likely continuation $f(p)$ based on the probability from the root to the current node $g(p)$ and the estimated cost of the most probable continuation to the leaf $h(p)$:

$$f(p) = g(p) + h(p) \tag{2.2}$$

Heuristics for estimating $g(p)$ and $h(p)$ will depend on the nature of the search problem.

Backtracking Search Different from the two latter search algorithms, a *Backtracking Search* will pursue only one search path at the time. However, if the path leads to a state from which there are no further legal transitions (a parse failure), the algorithm will revert to an earlier decision point, and choose another path from this point onwards. Hence, a backtracking search will execute the same number of computations as a deterministic search if a parse failure does not occur, except for the additional heuristics that allow the parser to resume parsing from an earlier parsing state. This strategy is further discussed in Chapter 6, where a backtracking approach customized for a transition-based HPSG parser is presented.

2.7 Deterministic and Near-Deterministic Parsers

We will present some prominent deterministic and near-deterministic parsers from the last decade for dependency grammar, CFG and HPSG. Most of the parsers are described as a transition-based parser. The transition system and data structure depend on the grammatical framework in question; key to the parsers is the *Oracle*, i.e. a classifier that determines the next parser action (i.e. the transition) of the transition-based parser.

2.7.1 Dependency Grammar

Kudo and Matsumoto (2002) introduced an efficient deterministic dependency parser for Japanese. Their parser outperformed previous probabilistic models with respect to accuracy and efficiency. Yamada and Matsumoto (2003) applied a similar method for English, and obtained a near-state-of-the-art accuracy compared to other non-deterministic parsers that were evaluated:

> We experimented with dependency trees converted from Penn Treebank data, and achieved over 90 % accuracy. Though the result is little worse than the most up-to-date phrase structure based parsers, it looks satisfactorily accurate considering that our parser uses no information from phrase structures. (Yamada and Matsumoto, 2003, p. 195)

They applied a shift-reduce parser with three parser transitions: *shift*, *left* and *right*. The parsing algorithm was essentially a two-stage operation where (1) the *contextual features* for the given state-of-the-parser was collected, and (2) the parser transition is predicted as a classification problem using an SVM (Support Vector Machine, Vapnik, 1995) model.

MaltParser MaltParser (Nivre et al., 2007) is arguably the most comprehensive platform for deterministic data-driven dependency parsing. The system can induce a parsing model from a treebank or parse new input using a pre-trained parsing model that can be downloaded separately. It assumes an input buffer consisting of the tokenized words and corresponding part-of-speech tags. MaltParser supports three families of parsing algorithms:

1. Nivre's algorithm (Nivre, 2003, 2004)
2. Covington's algorithm (Covington, 2001)
3. Stack algorithm, (Nivre, 2009; Nivre et al., 2009)

The *arc-eager* linear-time algorithm introduced by Nivre (2003) defined four parser actions:

1. Left-arc(r)
2. Right-arc(r)
3. Reduce
4. Shift

The Left/Right-arc operation is parameterized with the named label of the dependency relation. At each deductive parsing step, the shift-reduce parser will initiate one of the corresponding parser actions, and assign a dependency relation label (r) for the Left/Right-arc transitions. The prediction of the correct parser action is an instantiation of a *history-based*

feature model, as introduced by Black et al. (1992). The feature model combines local and global features such as part-of-speech annotation, lexical features and dependency type features.

MaltParser can employ any linear classifier in its machine learning model. In a deterministic configuration the oracle will predict the *1-best* transition for each classification choice, and the parser will pursue this path consistent with a greedy best-first search, as discussed in Section 2.6.1. MaltParser has been adapted to a range of different languages, including English, Swedish, German, Italian, Turkish, iter alia.

Near-Deterministic Dependency Parsers Whereas the MaltParser, presented above, is inherently deterministic, near-deterministic dependency parsing has also been explored through the use of beam search. This is exemplified by the parsers developed by Zhang and Nivre (2011) and Zhang and Clark (2008).

2.7.2 CFG

Sagae and Lavie (2005) presented a deterministic CFG parser based on the same classifier-based strategy that was pursued by MaltParser. Given that the search-space of constituent trees is larger than for dependency trees, the potential efficiency gain of deterministic parsing would be higher than for dependency parsing, but the margin of error would also be greater, since a deterministic CFG parser is likely to reach more decision points, with more alternative paths than the same sentence would have encountered in dependency parsing.

The overall architecture bears a strong resemblance to MaltParser, although the transition system and feature model is configured for CFG parsing. Instead of building a dependency graph, it produces a constituent tree structure of unary and binary branches. Although they do not reach the same precision/accuracy as 'classic' state-of-the-art parsers (e.g. Collins, 1997; Charniak, 2000; Charniak and Johnson, 2005), "the simplicity and efficiency of deterministic parsers make them attractive in a number of situations requiring fast, light-weight parsing, or parsing of large amounts of data." (Sagae and Lavie, 2005, p. 125)

2.7.3 CCG

Zhang and Clark (2011a) developed a shift-reduce algorithm for parsing *Combinatory Categorial Grammar* (CCG; Steedman, 2000). This parser applies a beam-search algorithm, and is thus not deterministic. As can be expected, the accuracy of the parser increases with the size of the beam-width, at the cost of higher running times. In the results reported in the paper, the beam size was set to 16, and no further reports were given on how

the parser would perform in a deterministic greedy best-first search. The parser applies a global linear model, which together with the beam-search approach does not make the parser faster than the C&C parser (Clark and Curran, 2007), which is a chart parser.

Zhang and Clark (2011a) report that using the standard development and test sets from CCGbank, the "shift-reduce parser gives a labeled F-measure of 85.53%, which is competitive with the 85.45% F-measure of the C&C parser on recovery of predicate-argument dependencies from CCGbank." Compared to the deterministic parsers described by Sagae and Lavie (2005); Nivre et al. (2007); Ninomiya et al. (2011) inter alios, the focal point of Zhang and Clark (2011a) is to utilize a rich global feature model which would be intractable within a conventional chart parsing scheme, and the attention to parsing times is less underscored. They conclude that:

> Considered in terms of the wider parsing problem, we have shown that state-of-the-art parsing results can be obtained using a global discriminative model, one of the few papers to do so without using a generative baseline as a feature. (Zhang and Clark, 2011a, p. 691)

2.7.4 HPSG

The increased complexity of unification-based grammars like HPSG could potentially entail a substantial increase in efficiency by parsing deterministically, but the inherent hard constraints of unification-based grammar could cause a high number of parse failures. Ninomiya et al. (2011, 2009) provide a deterministic shift-reduce parser for HPSG where this issue is addressed:

> Deterministic parsing is one solution that can achieve simple and fast parsing without the mechanisms of the packed parse forest by accurately choosing search paths. (Ninomiya et al., 2011, p. 331)

To mitigate the problem with parse failures, they suggest default unification (Copestake, 1993; Carpenter, 1993) by overwriting inconsistent constraints in the grammar, outlining a deterministic and a non-deterministic configuration. Default unification is desirable from a robustness perspective, and this will be further discussed in Section 7.1.2.

Ninomiya et al. (2011) evaluate their parser on Enju2.3ß (Miyao et al., 2004). In Section 7.2.1 we will compare this parser with our own parsing platform, since this parser has many of the methodological and strategic assumptions in common with our parsing system.

Backtracking Along with the greedy best-first configuration, the parser provides a simple non-deterministic strategy which can be used when deterministic parsing fails. They propose a backtracking approach that, upon

parsing failure, reverts the parser's state to a previous state, and chooses the second-best parser action, and continues parsing from this point (this would be analogue to traversing upwards in the search tree depicted in Figure 2.13). The backtracking point is chosen to be the state where the two highest-ranking parser actions have the smallest difference in probability.

Beam Search As an alternative non-deterministic approach, they apply a beam-search strategy where alternative parsing states are pushed to a priority queue based on the product of the probabilities for parser actions that were taken to reach the current state. However, to avoid penalizing longer search paths, a parse state will be added to the priority queue only if it is "greater than $\frac{1}{b}$ of the score of the best state in the states that had the same number of shift-reduce actions." (Ninomiya et al., 2011, p. 349) The accuracy of the parser is highly dependent on the beam width. With a wide beam they reach state-of-the-art accuracy, but at a low efficiency:

> The best-first parsing achieved the best *LF* of 89.9 per cent with an average parsing time of around 15 s. Even though this parser cannot be used in practice due to this efficiency, this indicates that our deterministic parsing has the potential to reach state-of-the-art parsers in terms of accuracy due to its truly deterministic nature and flexibility in feature design. (Ninomiya et al., 2011, p. 362)
> *LF=Labeled F-score*

Partially Deterministic HPSG parsing

Matsuzaki et al. (2007) present a fast, partially deterministic shift-reduce HPSG parser. The parser requires a preceding non-deterministic supertagging and CFG-filtering stage prior to the unification-based parsing which is done through a deterministic shift-reduce algorithm. This preprocessing effectively identifies and prunes a large number of ill-formed analyses, hence keeping the number of "'unfruitful' invocations of the HPSG parser" to a minimum (Matsuzaki et al., 2007, p. 1676). With this strategy, they achieve a significant speed-up compared to prior parsers on the same grammar, reporting a parsing speed of 30 msec/sentence, and keeping the accuracy at a comparable level.

2.8 Summary

This chapter presents the theoretical foundation for the subject of this thesis. We have included a short introduction to parsing, and contrasted chart parsing to deterministic approaches, paying particular attention to innovative transition-based deterministic and near-deterministic parsing algo-

rithms. As documented in this chapter, deterministic parsing has had a rapid development over the last decade, and we have seen research efforts for deterministic parsing for a broad range of grammatical formalisms. Since HPSG parsing, especially for the *English Resource Grammar* (ERG) (ERG will be introduced in Chapter 3), pertains to this project particularly, special emphasize has been given to initiatives within this framework.

We have given a short formal introduction to exhaustive and near-exhaustive chart-based parsing approaches. Deterministic and near-deterministic transition-based approaches have also been introduced, but will be further elaborated and exemplified in Chapter 5 and 6.

Chapter 3

ERG and DELPH-IN Resources

DELPH-IN (*Deep Linguistic Processing with HPSG Initiative*)[1] is a cooperation between academic institutions and researchers working with deep linguistic processing of human language. The consortium captures a range of tools and linguistic resources under its umbrella. In this chapter we will introduce the resources we have developed over the course of this project, and this project's interrelation to the resources provided by DELPH-IN.

The English Resource Grammar (ERG) provides the grammatical framework which serves as linguistic foundation for this thesis, and will be presented in Section 3.2. We have used two branches of annotated corpora: The Redwoods Treebank is a collection of treebanks of various domains which are manually disambiguated and grounded in the ERG. Secondly, WikiWoods is a large (approximately 47 million utterances) corpus annotated with ERG derivations by the PET HPSG Parser (see Section 2.5.3). As the transition-based parsing system we are developing does not have an explicit notion of grammatical rules, it learns the grammar through training on these annotated language resources. We will present these resources in Section 3.3.

Before we present the language resources, we will however clarify the distinction between manually disambiguated treebanks and automatically annotated (i.e. parsed) corpora (Section 3.1), which are terms we will use throughout the thesis.

3.1 Treebanks and Annotated Corpora

Treebanks are an indispensable resource for a wide range of disciplines within the NLP community. Formally, a treebank is a linguistically annotated collection of utterances. We distinguish between manually and semi-

[1] `http://www.delph-in.net/`

automatically annotated treebanks on one hand, and parsed corpora on the other. In manually and semi-automatically annotated treebanks every annotation is either handmade or manually corrected (from a parsed annotation), ensuring that the annotations are linguistically sound. An automatically annotated corpus is a corpus enriched with linguistic annotation from a parser which has not undergone manual correction, and is hence deemed to contain some portion of errors. An example of this is WikiWoods (Flickinger et al., 2010), which will be introduced in Section 3.4.3. We abstain from referring to automatically annotated, i.e. parsed corpora as treebanks in order to make the distinction between hand-corrected linguistic annotations and automatically annotated data clearer.

The creation of a large-scale treebank requires skilled linguists and a large amount of resources, several person-years of commitment is often required. The magnitude of the task is reflected in the relatively sparse selection of large-scale treebanks available. Penn Treebank (PTB; Marcus et al., 1993) has, since its release, been the dominating treebank for English, and has influenced data-driven research in NLP for two decades.

3.2 LinGO English Resource Grammar

There exists a variety of different HPSGs for a handful of languages. In this thesis we apply LinGO (Linguistic Grammars Online) ERG (English Resource Grammar; Flickinger 2000), which is a broad-coverage, linguistically precise handcrafted grammar for English. It is continuously being developed and expanded to increase coverage for new domains. The grammar is developed with the LKB (Linguistic Knowledge Building) grammar development environment – a multi-purpose system that can process ERG and other grammars that conform to a typed feature structure formalism (Copestake, 2002). The ERG provides a manually constructed lexicon of some 35,000 entries, designed to include all closed-class words of the language as well as most verbs of reasonable frequency and most of the syntactically idiosyncratic nouns, adjectives and adverbs.

Figure 3.1 presents an HPSG derivation for a simple sentence. This tree of ERG rules can be presented as a feature structure, and a semantic MRS (Minimal Recursion Semantics, Copestake et al., 2005) representation can be derived directly from its structure.

Figure 3.1: HPSG derivation from the WeScience Treebank.

3.2.1 Lexical Rules and Lexical Types

The derivation in Figure 3.1 is composed of lexical and syntactic rules. Further, each token is assigned a *lexical type* as preterminal (the concept of *lexical type* will be elaborated in Chapter 4). From the leaves in Figure 3.1 we see that punctuation is considered a part of the token, e.g. for *RSS-*. The lexical types end with the *_le* suffix, and we see that the lexical type for "RSS-", *n_-_pn_le*, provides the HPSG sign template for a proper noun. The sign is further augmented by the lexical ERG rules, which end with **lr*. The hyphen in 'RSS-' is hence annotated with *w_hyphen_plr*.

For 'specialized', the lexical type *v_np*_le* denotes the lexical category verb (*v*), with an optional noun phrase subcategorization argument. The lexical rule *v_pas_odlr* denotes a passive verb, and *v_j-nb-pas-tr_dlr* derives an attributive adjective from a transitive passive verb. Section 4.1.3 provides a more comprehensive introduction to the preterminals and lexical rules in ERG. Altogether there are 50 lexical rules in the ERG, and about 1,000 lexical types.

3.2.2 Syntactic Rules

Syntactic ERG rules have a *_c* suffix. The rules can be unary or binary. The root node in Figure 3.1, *sb-hd_mc_c*, denotes the conventional *Head+Subject* main clause in HPSG, in turn connecting a *Head+Specifier* (*sp-hd_n_c*) and a *Head+Complement* (*hd-cmp_u_c*) phrase in a binary production. There are in total 145 binary and 55 unary syntactic rules in the ERG.

3.2.3 Start Symbol

root_strict in Figure 3.1 constitutes the start symbol of the derivation. There are in total four start symbols used in the treebanks. These symbols are not an integrated part of the HPSG derivation, but describe sentential qualities of the HPSG derivation, and all of start symbols confirm that the derivation is well-formed according to the ERG.

3.3 Treebanks and Corpora for the ERG

The DELPH-IN consortium maintains two branches of corpora annotated with syntacto-semantic analyses conforming to the ERG. The Redwoods Treebank (Oepen et al., 2004b) and WikiWoods. In the following sections we will introduce the language resources, and their relation to this project.

3.3.1 LinGO Redwoods Treebank

The LinGO Redwoods Treebank is a manually disambiguated treebank with complete syntacto-semantic analyses. Since its release it has been continu-

ously expanded with new corpora and domains. In the release used in this project, it contains approximately 37,000 disambiguated sentences from various domains. The initial motivation for releasing a new treebank was the inherent limitations of the pre-existing publicly available resources:

> (i) each resource has chosen to focus on a single stratum of linguistic description, either topological (phrase structure) or tectogrammatical (dependency structure), (ii) the depth of linguistic information recorded in these treebanks is comparatively shallow (limited syntax, little or no semantics), (iii) the design and format of linguistic representation in the treebank hard-wires a small, predefined range of ways in which information can be extracted from the treebank, and (iv) representations in existing treebanks are static and over the (often decade-long) evolution of a large-scale treebank tend to fall behind advances in formal linguistics and grammatical representation. (Oepen et al., 2004b, p. 557)

Because the Redwoods Treebank is anchored to the ERG, the annotations will be updated for each new release of the ERG. This dynamic nature will also ensure that when the treebank is expanded with new corpora, its annotations will be consistent with the rest of the treebank.

3.4 WeScience

The WeScience initiative was launched in 2008 and is an ongoing effort to provide resources that enable eScience research and development in our own field, i.e. Natural Language Processing. Currently the focus has been on semantic parsing of Wikipedia articles. As a long-term goal the initiative wishes to expand into relating natural language semantics and taxonomic knowledge, for example in relation extraction or ontology learning applications. WeScience is now a joint effort between the University of Oslo, the Center for the Study of Language and Information,[2] the German Research Center for AI,[3] and Saarland University,[4] with partial funding from the University of Oslo, the Norwegian Open Research Archives,[5] and the Norwegian Metacenter for Computational Science.[6]

Over the course of this PhD project, two new corpora have been released under the WeScience umbrella. These corpora are derived from Wikipedia, thus addressing the accelerated interest for user-created content (or more

[2] http://lingo.stanford.edu/
[3] http://www.dfki.de/lt/
[4] http://www.coli.uni-saarland.de/
[5] http://www.ub.uit.no/wiki/openaccess/index.php/NORA
[6] http://www.notur.no/

popularly, Web 2.0). These resources constitute most of the statistical training data for the HPSG Parser that have been developed in the PhD project. Section 3.4.2 will give an introduction to the WeScience Treebank, a new treebank grounded in the Redwoods grammar-based annotation approach, consisting mainly of Wikipedia articles related to our own field, NLP. Section 3.4.3 provides an introduction to WikiWoods, which provides syntacto-semantic annotations for the full English Wikipedia. The WeScience Treebank can be seen as a hand-annotated subset of WikiWoods.

3.4.1 WeScience Corpus – Wikipedia as Domain

The *WeScience Corpus* (Ytrestøl et al., 2009) is a collection of 100 Wikipedia articles. The approximately 270,000 tokens in the corpus amount to about 14,000 sentences, distributed over 16 segments. Our decision to use Wikipedia is no coincidence. The term Web 2.0, normally used to describe the World Wide Web development where user-created content has become more dominant, is now well established. Moreover, typical Web 2.0 sites have a dominant position among the most visited web sites on the web, where Facebook, YouTube and Wikipedia appear as the most visited sites with user-created content.[7]

Wikipedia's editing process distinguishes it from most other documents that are available online. An article may have countless authors and editors. In April 2008, the English Wikipedia received 220,949 edits a day, with a total of 175,884 distinct editors that month. It is thus indicative of a noticeably dynamic community. When expanding NLP research into user-created content, taking advantage of Wikipedia is a natural choice. Wikipedia offers free downloadable databases of its content on a regular basis.[8]

Selection

We opted for a high degree of automatization when selecting the articles for the WeScience Corpus. Of the 2,543,723 articles available in the Wikipedia snapshot we applied,[9] a large number of articles will consist of only one or two sentences, hence not be suitable for a corpus whose goal is to represent natural running text.

We aimed to capture a domain dominated with articles familiar to our own research community, namely Computational Linguistics and Natural Language Processing (NLP). Shrinking the selection down to 100 articles was a procedure of defining constraints. As a starting point, we used the

[7] `http://www.google.com/adplanner/static/top1000`

[8] `http://en.wikipedia.org/wiki/Wikipedia:Database_download`

[9] This snapshot was made available July 2008 – statistics on the growth of Wikipedia articles are continuously updated on `http://en.wikipedia.org/wiki/Wikipedia:Modelling_Wikipedia's_growth`

inherent category labeling system of Wikipedia, which is not as well maintained as the articles. The category *Computational Linguistics* contained 355 articles. These articles were either categorized directly under *Computational Linguistics*, or under any of the subcategories for the category. We used these articles as our initial seed set, and performed a simple link analysis to count cross-references to other Wikipedia articles from the original seed articles. By taking the most frequently cross-referenced articles, and excluding very short articles, we were able to concentrate a ranked list of hopefully the most relevant articles for our domain. See Ytrestøl et al. (2009) for a more comprehensive presentation of the article selection stage.

Preprocessing and Format

Wikipedia articles are edited in *Wiki Markup Syntax*, a markup language that facilitates on-line rendering (as HTML) for display in a web browser. For the WeScience Corpus it was necessary to purge the articles of non-linguistic content and irrelevant markup in order to be able to maintain a suitable body of text for further linguistic processing. Irrelevant markup, e.g. in-text image data, was removed through a cascade of regular expression, and the text was formatted in our own native line-oriented WeScience exchange format.

For sentence segmentation we used *Tokenizer*,[10] a rule-based tool which proved very capable as a sentence segmenter. Its performance was further enhanced by additional regular expressions. The WeScience format presents one sentence per line, and each line has a unique eight-digit sentence identifier. The sentences are consecutively distributed across 16 sections, where each section comprises up to 1,000 sentences, and no articles are split between two sections.

3.4.2 WeScience Treebank

The WeScience Treebank is an annotated subset of the WeScience Corpus featuring gold standard HPSG analyses. The first 13 segments have, since September 2009, been available in the Redwoods Treebank collection, with the complete syntacto-semantic analyses that were provided by the ERG in accordance with the rest of the Redwoods Treebank (see Section 3.3.1). The WeScience Treebank features 9,100 gold standard analyses from the first 13 sections of the WeScience Corpus.

Treebanking

The corpus has undergone manual correction and disambiguation using the open-source LinGO Redwoods environment, based on the ERG. This tree-

[10] `http://www.cis.uni-muenchen.de/~wastl/misc/`

banking served multiple purposes. It provided (a) higher-quality annotation in general, (b) an estimate of the expected error rate in the automatically assigned annotation and (c) provided training data which can be used for statistical parse disambiguation.

To the best of our knowledge, ERG has not been applied to Wikipedia text prior to this initiative, and we were using pre-existing out-of-domain parsing models for initial parsing. We recorded all of the candidate analyses (up to a practical limit of 500 per utterance), and employed the *[incr tsdb()]* Redwoods platform for grammar competence and performance profiling (Oepen et al., 2004b), which includes a graphical tool for the task of disambiguation, enabling annotators to identify the intended analysis among the candidates in the parse forest. If the correct analysis did not appear among the top 500 analyses provided by the parser, the sentence was rejected from the treebank. Hence, the treebank can be regarded the subset of sentences in the WeScience Corpus (1) that were grammatical according to ERG, and (2) where the PET Parser was able to process and retrieve the correct analysis. Parsing errors most frequently occurred due to "grammar shortcommings, e.g. pseudopassives (*... is referred to as*)" or "parse ranking: good analysis likely available, but not in top 500" (Ytrestøl et al., 2009, p. 194). See Ytrestøl et al. (2009) for a more comprehensive discussion on the treebanking process.

WeScience Derivation The syntacto-semantic analyses provided by ERG can be exported to various kinds of linguistic representations at a variable level of granularity, as seen in Figure 3.2 and 3.3. Figure 3.2 (top) provides the full HPSG analysis, formatted in a derivation tree. In Figure 3.2 (bottom) the HPSG representation is simplified to a phrase structure tree.

Figure 3.3 provides the MRS logical form of the same sentence, currently not scope resolved. The representation provides three types of logical variables: events (e_i), instances (x_j) and handles (h_k). MRS and the semantic representation of an ERG analysis will be further discussed in Section 7.1.2, and we refer to Copestake et al. (2005) for a comprehensive presentation of Minimal Recursion Semantics.

3.4.3 WikiWoods

WikiWoods[11] is a parsed corpus consisting of approximately 1.3 million Wikipedia articles. All articles are extracted from the Wikipedia snapshot dated July 2008, i.e. the same that was used for WeScience. While the Wikipedia dump contains over seven million article elements in total, the majority of these are redirects or have non-encyclopedic content (e.g. help pages and Wikipedia-internal discussion threads). For WikiWoods we

[11] WikiWoods will in near future be branded as *WikiWoods Treecache*.

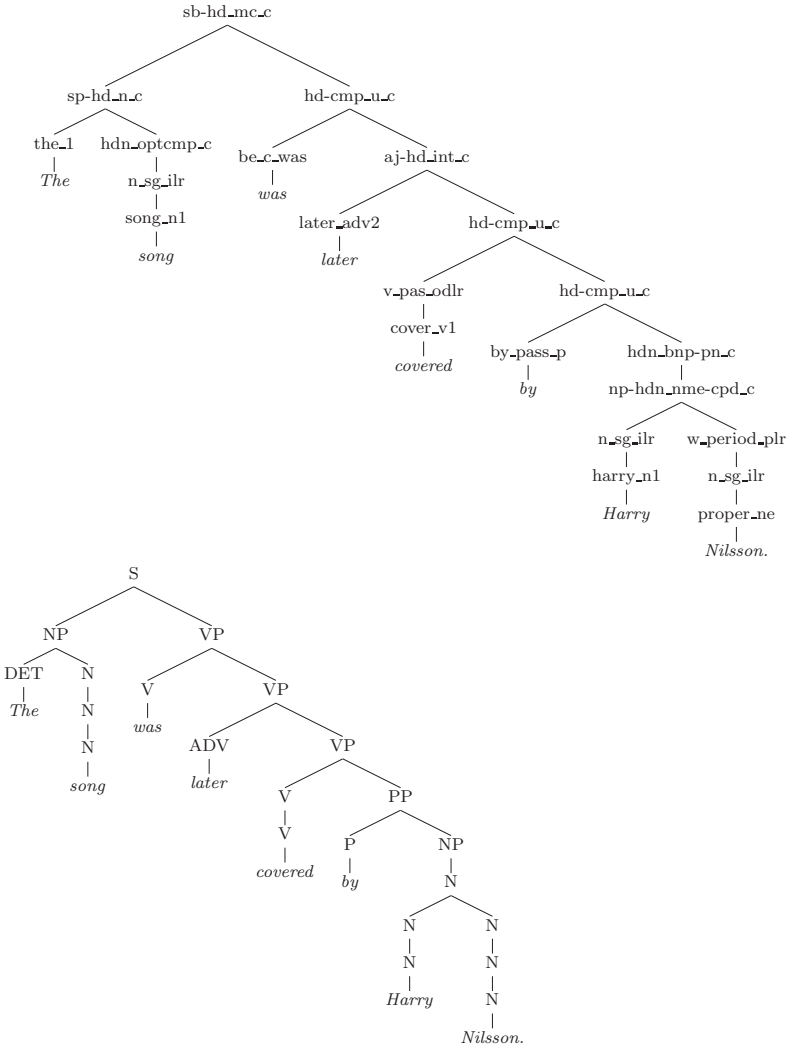

Figure 3.2: Syntactic representations for *The song was later covered by Harry Nilsson.* The HPSG derivation (above) is labeled with identifiers of lexical entries and constructions; the phrase structure tree (below) reduces HPSG signs to conventional category labels.

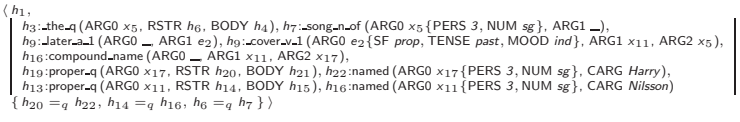

Figure 3.3: Semantic representation (compare to Figure 3.2). The details of underspecification are not important here, but note that the arguments of the passive are adequately recovered.

excluded these articles, as well as any article with less than 2,000 characters, opting for extracting well-maintained coherent running text. Having applied these constraints, we were left with 47 million utterances, with an average length of 15.4 tokens per utterance.

Automatic Annotation

For the automatic annotation process, we used the PET HPSG Parser (see Section 2.5.3). Having already treebanked a Wikipedia subdomain, we could train a statistical model for parse ranking on the WeScience Treebank. For each utterence, the PET parser would construct the full parse forest licensed by ERG, and the highest ranked parse derivation would be recorded without any further manual inspection. We reached a grammatical coverage of just below 85%, which is comparable to, though slightly lower than the earlier WeScience experiment. For the remaining sentences, parsing failed, and the sentences were excluded. The cause for this was either that there was no valid reading of the utterance licensed by ERG, or because the parser timed out, something that happened if constructing the parse forest took more than 60 seconds.

Coverage and Annotation Quality

In order to obtain a rough assessment of the performance of the statistical parse disambiguation, we randomly selected 1,000 parsed sentences from the WikiWoods, and carried out a manual evaluation of the quality of the top-ranked analyses assigned by the grammar. Excluding a handful of items which suffered from incorrect sentence segmentation or typographical errors, each parse was classified to be of one of three qualities: *correct*, *nearly correct*, or *incorrect*.

The preliminary manual evaluation suggests that the quality of the automatically assigned analyses is quite good, with more than 83% of the analyses judged as correct or nearly correct, and more than two-thirds judged as fully correct. As can be expected, the percentage of incorrectly analyzed items rises as the sentence length increases, but even for the sentences of 5 to 24 tokens in length, 82.6% received a nearly or fully correct analysis.

Flickinger et al. (2010) provide a more extensive report on the creation and quality of WikiWoods.

3.5 Data Sets Used in the Thesis

The data-driven system for HPSG parsing developed in this project relies heavily on annotated training data. The total number of sentences in the gold standard Redwoods Treebank is approximately 37,000, which proved insufficient for many of the training models that were used in this project. Most notably, the supertagger (introduced in Chapter 4) benefited from an amount of training data that was several orders of magnitude larger than the size of the Redwoods Treebank. Further, the near-deterministic backtracking overlay which will be presented in Chapter 6 required training data which had not been used for training the parsing oracle (this will be explained in Section 6.5.2). It became clear from an early stage that the gold standard Redwoods Treebank did not supply a sufficient amount of training derivations for all aspects of this project. The availability of WikiWoods provided, with its 47 million derivations, more than enough annotated data for this project, and without this source it is likely that we would have resorted to resample additional training data.

We structured the annotated corpora and treebanks in two sets for this project: Redwoods Treebank, and the automatically annotated WikiWoods corpus (see Section 3.2 for more details on the Redwoods Treebank and WikiWoods). The Redwoods Treebank contains data from a range of domains. Table 3.1 gives an overview of the corpora from the Redwoods Treebank that are used throughout this thesis. From the WikiWoods corpus we applied data from segment 2000-4000, in total approximately 10 million sentences (see Table 3.2).

In Table 3.1 we see that the Verbmobil (*vm*) corpus is the largest Redwoods data set in terms of number of sentences. The Verbmobil corpus (Wahlster, 2000) consists of transcribed dialogues, which explains the relatively short average length per utterance. The *jhpstg* corpus contributes with the largest amount of tokens, and derives its input from translated tourist guides used in association with the LOGON project (Oepen et al., 2004a). SemCor (*sc*) is a semantically annotated subset of the English Brown Corpus (Francis, 1979; Francis and Kučera, 1982). Additionally, we use the WeScience Treebank (Ytrestøl et al., 2009), section 1-11 for training, section 12 for development and section 13 for testing (see section 3.4 for more details on WeScience).

From this overview we can establish that the Redwoods Treebank is neither domain nor genre specific, but given that approximately 30 % of the data is derived from WeScience we can say that WeScience section 13 is partially in-domain as test data. Further, for out-of-domain test data

Corpus Name	# of Sent	Avg # Tok/Sent	# Tok	Usage
WeScience 1-11	7,636	14.4	109,600	train
WeScience 12	601	15.6	9,400	dev
jhpstg	8,501	13.3	113,000	train
Verbmobil	11,116	6.8	75,600	train
SemCor	2,564	15.1	38,700	train
ecpr	1104	9.4	10,426	dev
The Cath. and the Bazaar	570	18.4	10,500	test
WeScience 13	785	14.1	11,100	test

Table 3.1: The corpora in the Redwoods Treebank that are used as training and test data in this project.

Corpus Name	Segment	# of Sent	Avg # Tok/Sent	# Tok	Usage
WikiWoods	2000-4000	10.3 mill	13.8	142 mill	train

Table 3.2: The WikiWoods corpus is used to augment the training data and provides automatically annotated derivations. In this project we applied segment 2000-4000, approximately 20 % of the entire WikiWoods corpus.

sets we included the technical essay *The Cathedral and the Bazaar*[12] (*cb*), and the eCommerce corpus *ecpr* serves as an out-of-domain development set. The annotated data presented here is the full set of training and testing resources we have used for this project, both with respect to the supertagger (introduced in Chapter 4), the deterministic HPSG parser (see Chapter 5) and the near-deterministic backtracking overlay (introduced in Chapter 6).

3.6 Conclusion

The system developed in this project relies on statistical training data in multiple stages. Since it can operate without an explicit notion of grammar rules, it learns the grammar through the training derivations. The language resources presented in this chapter will be the backbone for all our statistical models in this thesis. The extraction of the WeScience Corpus demonstrates a simple and efficient approach to acquiring a large-scale corpus, a method which could be implemented to any language that has a reasonable amount of Wikipedia articles.

The release of WikiWoods provides disambiguated annotations for 47 million Wikipedia sentences. We have for this project not utilized the full scale of this parsed corpus, but we expect that future research efforts will be even more committed to advancing the performance of statistical models through large-scale data processing. Both in research and in industry, the

[12] `http://catb.org/esr/writings/cathedral-bazaar/`

use of distributed systems for data-intensive processing of large data sets has surged through the use of frameworks like Hadoop[13] and MapReduce (Dean and Ghemawat, 2008). We are confident that automatically annotated corpora like WikiWoods could be an enabler for other large-scale text processing initiatives, e.g. for efforts within parsing, machine translation, data mining and information retrieval.

[13] http://hadoop.apache.org/

Chapter 4

Supertagging

This project introduces a transition-based parsing platform for HPSG. The parser has been designed to assume an explicit input buffer for each sentence, and this input buffer is generated in a preprocessing stage. In this chapter we will present this preprocessing stage, with a focus on the most crucial aspect, namely the *supertagging*. We will further document the design choices we made, and evaluate how the performance of our supertagger compares to previous taggers within this domain.

The fundamental idea behind supertagging derives from an intuition that has been discovered and rediscovered throughout cultures and time, namely that words can be clustered together with other words based on their properties. Although there is a considerable amount of controversy regarding the universalness of part-of-speech (POS) categories, there is strong evidence that each individual language can structure the words of the language into classes based on the grammatical properties of the words.

Dionysius Thrax (100 B.C.) classified the words of Greek into eight categories: Noun, verb, pronoun, preposition, adverb, conjunction, participle and article. Although outdated in modern linguistics, the categories have maintained their position as the most popular classification scheme for lexical categories in Western languages for more than 2000 years. The need for more fine-grained generalizations within modern linguistics has lead to a significant increase in the number of lexical categories (conventionally called *tags*). For English, Penn Treebank (Marcus et al., 1993) applies a tagset of 45 lexical categories, the Brown corpus (Francis, 1979; Francis and Kučera, 1982) has 87 and the CLAWS C8 tagset consists of 170 tags (Garside and Smith, 1997).

In constituent-structure trees, the *preterminal* is the term used to describe the node that dominates the terminal. Preterminals, i.e. categories that generalize over the words in the lexicon, are essential to any linguistic framework. However, each framework has different requirements that call for a different set of lexical categories. It is a sliding scale where we can

have a small tagset that captures a high degree of generalization, or a larger tagset with more descriptive power for each tag. For a lexicalized grammatical framework, conventional POS tags would typically fail to capture many of the more subtle inherent lexical constraints that each word embodies.

Supertags were introduced to capture a finer-grained generalization of words than the POS tagsets provided, to meet the needs of parsing systems in a lexicalized grammar (more on this in section 4.1 below). Finer-grained preterminals will lead to a narrower search-space with a reduced complexity in a subsequent parsing stage. Supertags are generally closely integrated with a parser, providing the parser with preterminals that are optimized for the grammar and parsing platform being used. Hence, state-of-the-art supertaggers are generally built specifically to provide a certain parser with input data.

4.1 Supertags

The term *supertag* was introduced by Joshi and Srinivas (1994) in the paper *Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing*. Indeed, *Almost Parsing* seems to be a descriptive two word summary of what supertagging is. When supertagging a sentence, each lexical item is assigned a rich description that will "impose complex constraints in a local context" (Bangalore and Joshi, 1999, p. 237). Part-of-speech tags (POS tags) had been used for similar purposes, namely to categorize and group lexical items according to their use in a local context. However, conventional POS tags were deemed too broad to fit the lexical item description defined by the LTAG (Lexicalized Tree-Adjoining Grammar, Joshi and Srinivas, 1994; Joshi, 1987) elementary trees, which made up the lexical framework that was the subject for their research. LTAG elementary trees define a complex lexical item description, designed to "allow for all and only those elements on which the lexical item imposes constraints to be within the same description."(Bangalore and Joshi, 1999, p. 237) Originally developed for the LTAG framework, Bangalore and Joshi (1999) claim that after an initial supertagging of a sentence, the search space of the remaining syntactic parsing is shrunk to such an extent that this task becomes almost trivial, given that the LTAG elementary trees capture so much information and so many constraints in the lexical items. In their original work, they assumed a tagset of 300 elementary trees.

4.1.1 Tags and Granularity

While the supertag in Bangalore and Joshi (1999) was defined through a specific linguistic entity of the LTAG framework, namely the elementary trees, the intuitions of a supertag was soon conveyed to other frameworks. Hence, multiple alternative supertag definitions have emerged, capturing various

degrees of lexical information. Maintaining a clear distinction between POS tags and supertags is complicated, and we will for practical purposes not assume supertags to be inherently different from POS tags in more than one way: Supertags capture more fine-grained linguistic information than what POS tags conventionally do.

This leads us to the various linguistic properties that can be captured by a POS tag or supertag. We do by no means intend to present an exhaustive list of such properties, but focus on the aspects that are especially relevant for this project.

Major Category The *major category* corresponds to the traditional set of *lexical categories*, namely the noun, verb, pronoun, preposition, adverb, conjunction, participle and article. Minor variations and extensions may occur. We assume that most tagsets would minimally include this information.

Morphology Morphologic features refer to inflectional variations of a lexeme. This could imply making a distinction between singular and plural nouns (*car – cars*), or between a third person singular verb form *drives* and the non-third person singular form *drive*. This information is captured in the PTB tagset, with *NN* and *NNS* tags distinguishing singular and plural nouns, and *VBZ* and *VBP* distinguishing third person present and non-third person present verbs.

Subcategorization The tag can be further augmented with the subcategorization frame of the token. This describes which type of specifiers or complements the token allows, and would hence define an explicit distinction between an intransitive and a transitive verb. This information is typically not captured in traditional POS tagsets like PTB. Such information is however specified in a LTAG elementary trees, and is hence captured by the supertags assumed by Bangalore and Joshi (1999).

Selectional Relation While the subcategorization frame would annotate the token's properties regarding specifiers or compliments, the selectional relation can further tighten the constraints of the subcategorization frame. Specifically, this can be achieved by explicitly stating the word or words that may appear in the given subcategorization frame. From the ERG (ERG was introduced in Chapter 3, and supertagging for ERG will be elaborated in Section 4.1.3) we find an example of this for the verb *argue*, which is represented by different supertags depending on the head of the prepositional phrase it could take as a complement. Hence, we have alternative supertags for 'argue+about', 'argue+for' and 'argue+ over'.

Semantic Linking Through semantic linking the syntactic argument is linked to the semantic role in a verb relation. This is typically relevant for *unaccusative verbs*, verbs whose syntactic subject does not correspond to the semantic agent. In the sentence

(1) The window was closed by Sandy.

we see *closed* in an unaccusative relation, where the semantic agent is *Sandy*, even though the syntactic subject is *The window*.

4.1.2 The Supertag in HPSG

While the elementary trees in the LTAG framework naturally correspond to the concept of a supertag, the *lexical categories* take on the same role within CCG (Clark and Curran, 2004). However, the HPSG formalism leaves more flexibility when defining what the supertag should be. The tagset for HPSG supertags varies from grammar to grammar, and even within research projects working on the same grammar, because there is not one single linguistic entity within HPSG that relates directly to the concept of a supertag as defined by Bangalore and Joshi (1999).

An HPSG supertag will relate to the HPSG word sign. As we discussed in Chapter 2, an HPSG lexical sign captures information on the constraints that the lexical item imposes on the surrounding lexical items in the sentence, together with properties of the lexical item, such as *gender*, *number*, or possibly more subtle features like semantic linking. As mentioned in Section 2.4, the initial grammars for HPSG contained only a small set of phrasal rules, something which would imply that subsequent parsing would be much simpler as long as the supertags that approximate word signs are assigned.

However, as mentioned in Section 3.2, modern large-scale grammars for HPSG often have a much more complex grammar – this is exemplified by the ERG, which incorporates 200 phrasal rules. An initial supertagging would shrink the search-space substantially, but nowhere near the point where one could claim that the remaining syntactic parsing is trivial.

For the Enju Grammar, Ninomiya et al. (2006) assume a tagset of 1,361 supertags, using the lexical templates in the grammar as tags (section 2.5.4 gives a brief introduction to the Enju Grammar). This tagset is also used by Matsuzaki et al. (2007).

4.1.3 Supertagging in ERG

The decision of what linguistic granularity a supertag within ERG should capture is ultimately a design choice that should be made in consideration with the overall use case. After all, supertagging in itself typically serves as a stage in a pipeline of processing tasks, aiming to boost overall performance of a system. For this project the supertags will serve as input for

the transition-based deterministic HPSG Parser, CuteForce, which will be introduced in Chapter 5. For the HPSG parser, a preprocessing stage entailing supertagging is in itself not a requirement, one could easily imagine that all preterminals were assigned by the deterministic parser. There are however strong indications that this would be an inferior solution compared to allowing a preprocessing stage for this purpose, as we will discuss in Section 5.2.1.

The scope of our task bears similarities to the project that was presented in Dridan (2009). In this thesis Dridan evaluated various lexical properties that could be exhibited by the supertag, from a simple tagset of 13 POS tags, up to "very fine-grained descriptions that include information about morphology, subcategorisation and selectional preference of prepositions." (Dridan, 2009, p. 6) We will assume the same terminology as used by Dridan (2009), and will present this below. We will return to prior research on supertagging in ERG in Section 4.1.4.

Lexical Entry The lexicon of the grammar is populated with lexical entries, and version 1010 holds some 35,000 lexical entries in the ERG lexicon. The lexical entry consists of (1) a unique identifier, (2) a lexical type, (3) a stem, (4) an optional selectional relation, (5) a semantic predicate and (6) phonetic information (see Figure 4.1). Within the LKEYS attribute we find the semantic predicates that will be used in the semantic analysis. PHON.ONSET and STEM relate to the written/spoken form. Each lexical entry is assigned exactly one *lexical type*, which will be discussed below.

(1)think_of := (2)v_pp_e_le &

$$\left[\begin{array}{ll} \text{ORTH} & \left\langle (3)\text{"think"} \right\rangle \\ \text{SYNSEM} & \left[\begin{array}{ll} \text{LKEYS} & \left[\begin{array}{l} \text{–COMPKEY (4)\_of\_p\_sel\_rel} \\ \text{KEYREL.PRED (5)"\_think\_v\_of\_rel"} \end{array}\right] \\ \text{PHON.ONSET} & \text{(6) con} \end{array}\right] \end{array}\right]$$

Figure 4.1: Lexical Entry for *think_of* from the ERG.

Lexical Type The lexical type is a lexical template prototypically generalized as a feature structure within HPSG. It contains category information such as the head type and the subcategorization fields of the lexical entry. There are about 1,000 lexical types in the current version of ERG (release 1010). The name of a lexical type is structured as a pattern that holds a major lexical category field, a subcategorization field and an optional descrip-

tion field of the form <lex-cat>_<subcategorization>_<description>_le.[1] Hence, the lexical type v_pp_e_le describes a lexical entity with a **v**erbal lexical category field and a **pp** complement in the subcategorization field and an **e**mpty description field. The actual name does not autonomously reflect the full linguistic definition that the lexical type captures – the complete description is stored as a feature structure for each individual lexical type. Figure 4.2 is a simplified feature structure for the information associated with the lexical type *v_pp_e_le*. We see that the valency constraints are captured within the VAL attribute.

v_pp_e_le

$$\left[\begin{array}{l}\text{SYNSEM}\left[\begin{array}{l}\text{LOCAL}\left[\begin{array}{l}\text{CAT}\left[\begin{array}{ll}\text{HEAD} & \text{verb}\\ \text{VAL} & \left[\begin{array}{ll}\text{SUBJ} & \left\langle\left[\text{LOCAL}\left[\text{CAT nomp\_cat}\right]\right]\right\rangle\\ \text{SPR} & \langle\rangle\\ \text{COMP} & \left\langle\left[\begin{array}{l}\text{LOCAL}\left[\text{CAT basic\_pp}\right]\\ \text{-MIN } \boxed{1}\text{selected\_rel}\end{array}\right]\right\rangle\end{array}\right]\end{array}\right]\end{array}\right]\\ \text{LKEYS}\left[\text{-COMPKEY } \boxed{1}\right]\end{array}\right]\end{array}\right]$$

Figure 4.2: Simplified feature structure for the lexical type *v_pp_e_le*.

Lexical Item The HPSG word sign within ERG is referred to as the *lexical item*. In practice, the lexical item consists of a lexical type augmented with one or more lexical rules. Figure 4.3 shows the lexical inflection rules for the token *"population,"*.

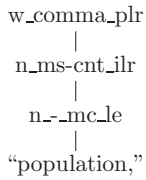

Figure 4.3: Lexical item for *population,*. The lexical type *n_-_mc_le* is augmented with a lexical inflection rule (*n_ms-cnt_ilr*) and a punctuation suffix rule (*w_comma_plr*).

[1] `http://moin.delph-in.net/ErgLeTypes` provides a more comprehensive description of lexical entry conventions.

During parsing these morphological or punctuational inflection rules will augment the lexical type, as illustrated in Figure 4.3. Since a token can (in principle) be followed by any number of inflection rules (for instance for the token 'HELLO' followed by an infinite number of exclamation marks), there is no upper limit to the number of alternative lexical items the grammar can produce.

4.1.4 Previous Supertag Definitions for ERG

Blunsom (2007) presented and evaluated supertagging models for HPSG and CCG. For the English HPSG model, Blunsom used ERG and the Redwoods treebank. As tagset the ERG lexical types were applied, as introduced above. At this time, the number of unique lexical types instantiated in the treebanks were 615, from a total number of 915 lexical types in the grammar. Applying a CRF learner, a training model from 10,000 sentences from the Redwoods treebank was built.

Toutanova et al. (2002) described an approach for HPSG parse disambiguation for the ERG. In their work they built a tagger for the lexical entries in the treebank. Contrary to Blunsom (2007), this constitutes a much more fine-grained tagset of 8,000 tags, which was the total number of lexical entries[2] in the ERG treebanks at that time.

Additionally, Dridan (2009) presented a comprehensive evaluation of a wide range of supertags of diverse granularities. This project aims to discover, among others, how supertagging of textual input could be of benefit to the PET Parser in terms of accuracy, coverage and speed when parsing with the ERG. By providing the PET parser with supertag preterminals for each input token, thus imposing lexical restrictions, the search space of the parser is reduced because fewer preterminals are added to the chart. Hence, Dridan (2009) opted to strike the best possible balance between a linguistically rich supertag that would reduce the parsers search space, and the supertagger's ability to keep a high accuracy.

> While we expect that a *useful* tag set will in fact make linguistically meaningful distinctions, which distinctions are useful will depend on the make up of the parser and grammar. (Dridan, 2009, p. 64)

Eight different tagsets of variable granularity were evaluated for this specific task. The most complex tagset was the lexical type, augmented with morphological information and selectional relation (following the definitions introduced in Section 4.1.1), in total 1217 different tags in the training data. Contrastively, the simplest tagset consisted only of the major lexical categories, in total different 13 tags. The other tagsets are combinations

[2]The term *lexical label* is used in the paper by Toutanova et al. (2002).

of linguistic information such as major categories, subcategorization frame, morphology and selectional relations. As can be expected, the accuracy of the tagger largely depended on the granularity of the tags, where tags providing finer linguistic information are more challenging to tag accurately. The optimal configuration in terms of lexical restriction depends on the priorities for the parser. If speed had a high priority, the *lexical type* offered the best coverage and accuracy scores. However, if speed was less underscored, a supertag of a coarser linguistic granularity would yield a better coverage.

4.1.5 The Supertag in this Project

Based on the experiences made in Dridan (2009), and the overall requirements of the deterministic parser, we decided to apply the *lexical type* as defined above (see Section 4.1.3) as the supertag input for our parser. We find that this level of granularity will provide the best possible balance between lexical restriction of the search-space and the supertagger's potential to achieve a high tag accuracy, and also maintain the best possible correspondence between the conventional ERG representation, and the derivations produced by our parsing system.

4.2 Tokenization

To *tokenize* a text is, generally speaking, the task of extracting the individual elements that comprise a sequence of text. However, the definition of these fundamental components can be a matter of controversy, which has given rise to alternative tokenization regimes. For English, the dominating position of the Penn Treebank has manifested itself in the widespread use of PTB-like tokenization within the community. Parsers using grammar, or lexical descriptions derived from PTB (Ninomiya et al., 2011; Clark and Curran, 2007; Miyao et al., 2004, inter alios) will subscribe to this tokenization regime. There exist a number of tools for segmenting raw text according to a PTB-like tokenization regime. Although moderately simple regular expression-based heuristics can achieve a relatively high level of precision, there have however been reports of inconsistencies between different PTB tokenizers for domains outside the Penn Treebank. E.g. Velldal et al. (2010) reported tokenization inconsistencies in 20 % of the sentences for the BioScope corpus (Szarvas et al., 2008) when using the GENIA tagger (Tsuruoka et al., 2005).

The ERG assumes an alternative conception of a token compared to PTB. Maybe most striking when comparing the ERG and PTB tokenization, is that the ERG does not tokenize punctuation separately, but rather treats punctuation as pseudo-affixes to a word (except from colon and dashes). The punctuation affixes are treated similarly to lexical inflectional and derivational rules in a hierarchical level above the preterminals. Further, con-

tracted negations (like *don't*) are treated as a single token in ERG, whereas it is split into two by PTB (*do*, *n't*), as we see in Figure 4.4. Further examples of punctuational differences in ERG and PTB are presented in Section 4.3.1.

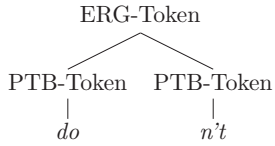

Figure 4.4: Contracted negations are tokenized as separate tokens in PTB, but treated as a single lexical units in ERG.

In addition to the punctuation regime, ERG diverges from PTB with the concept of multiword expressions (MWEs). This will be elaborated in the following section.

4.2.1 Multiword Expressions

> There are vast numbers of such memorized fixed expressions; these extremely crude estimates suggest that their number is of about the same order of magnitude as the single words of the vocabulary. (Jackendoff, 1997, p. 156)

A *Multiword Expression* (MWE), informally 'words with spaces', typically refers to two or more tokens that represent one syntacto-semantic unit. This could be idioms or habitual collocations like 'right off the bat' or 'beat around the bush' or phrasal verbs like 'look(ing) forward to'. Sag et al. (2002) define a MWE as "idiosyncratic interpretations that cross word boundaries (or spaces)". Further, they distinguish between *lexicalized phrases* and *institutionalized phrases*. A lexicalized phrase can be further broken down to *fixed expressions*, *semi-fixed expressions* and *syntactically-flexible expressions*. This subcategorization refers to the lexical rigidness of the 'word' items that constitute the phrase. Typically, we also observe that the more fixed an expression is, the more the expression behaves like a single word. Such fixed expressions, like 'ad hoc' and 'in short', cannot undergo any lexical variation.

In the other end of the scale we find syntactically-flexible expressions. These are phrases consisting of words that, when acting together as a unit, take on a different meaning than what the words in themselves would imply. Here we find (2) *verb-particle constructions*, (3) *decomposable idioms* and (4) *light verb constructions*.

(2) Brush up on.

(3) Let the cat out of the bag.

(4) Make a mistake./*Do a mistake.

We refer to Sag et al. (2002) for a further elaboration of these terms, and for a comprehensive introduction to multiword expressions, primarily in relation to linguistically precise NLP.

In ERG, there are roughly 3,000 lexicalized MWEs. Within ERG the identification of MWEs is done as part of the parsing stage, which is natural since a naive extraction of word collocations from a lexicon of MWEs would fail to recognize the difference between the MWE in (5) and the same collocation in (6), which is not an MWE.

(5) He is *easy going*.

(6) It is easy going there.

The supertagger we present in this chapter will treat an MWE as a single lexeme, and assign it a lexical type, supertag, accordingly. This strategy is discussed in Section 4.2.2. Although we assume the gold standard ERG tokenization as input for our HPSG parser, we need to recognize and format MWEs in the input so that they conform to PTB tokenization, because the POS tagger assumes this input. This will be further discussed in Section 4.3.1.

4.2.2 Tokenization in This Project

For the training and test data used in this project we will assume gold standard tokenization as it is provided by the treebanks. This is parallel to the assumptions in Zhang and Krieger (2011), which describes a parser that is trained and tested on the ERG treebanks. This is also a common strategy for a number of parsers that use Penn Treebank for training and testing (Ninomiya et al., 2011; Matsuzaki et al., 2007; Sagae and Lavie, 2005, inter alios).

Alternatively, the tokenization could be performed as an integrated part of the preprocessing stage that converts running text to an input buffer of tokenized words, POS tags and supertags (this is also discussed in Section 8.4.4). The latter strategy was applied in Dridan (2009), and we refer to this work for a comprehensive presentation of a feasible pipeline from running text to a buffer of tokenized input equipped with ERG supertags.

4.3 Developing the Supertagger

The purpose of supertagging in this project is to provide CuteForce with a sentence input buffer, consisting of the word token, a part of speech (POS) tag (discussed in Section 4.3.1), and a supertag. Although the POS tags are

not part of the HPSG analysis, they are embedded in the feature model used by the discriminative parsing model. They are also applied by the supertaggers (presented in Section 4.3.3), which assume a list of ⟨Word Token/POS Tag⟩ tuples as input. Hence, prior to supertagging we will tag the input with POS tags.

4.3.1 Part-of-Speech Tagging

The POS tags are essential to the feature model of both the supertagger and the CuteForce Parser. Both the tokens of the training data and the test sets will be assigned a POS tag. The decision of tagging the training data requires some words of explanation. After all, one could imagine extracting gold standard POS tags from the treebank. However, this would lead to an inconsistency between the training data and the test data, which would be unfortunate for any classification task. Since the POS tags are not a part of the actual HPSG derivation, the POS tags will only contribute indirectly to the supertagger and the HPSG Parser in terms of providing features for the machine learning model. That the generalizations and assumptions in the training data do not differ from the test data, seems much more important than reducing the number of (possible) linguistic inaccuracies in the training data, because that would directly lead to more inconsistencies between the test and the training data.

POS Tokenization

As mentioned in Section 4.2, the tokenization regime in ERG differs from PTB. For POS tagging we apply the TnT tagger using the pretrained Wall Street Journal model which is released with TnT (Brants, 2000). Using a native ERG tokenization as input for this POS tagger would result in a large number of unknown words related to punctuation affixes and multiwords. Hence, we opted to convert the ERG tokenization to a PTB-like tokenization for the purpose of POS tagging. Specifically this means handling cases where the ERG tokenization would pose a particular problem to the TnT Tagger.

Punctuation Affixes ERG generally treats punctuation as pseudo-affixes, whereas PTB would normally treat them as separate tokens. Punctuation affixes in ERG trigger lexical morphological rules, similar to a so-called orthographemic inflection, see Figure 4.5.

The full expansion of the lexical rules and the punctuation affixation rules for the sentence *In fact, a well-defined end-state.* is shown in Figure 4.5. The punctuation affixes are labeled as lexical rules whose names start in *w*. For the PET parser these rules are selected from candidate lexical rules that are derived in an initial preprocessing stage. Such lexical punctuation

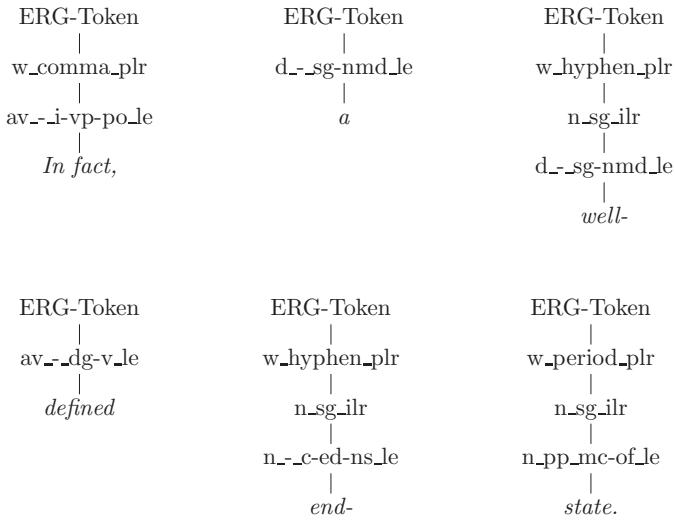

Figure 4.5: Lexical rules and punctuation affixation rules for the sentence *In fact, a well-defined end-state.*

rules allow for a simple token expansion where the punctuation affixes are identified and split up from the word token, see Figure 4.6.

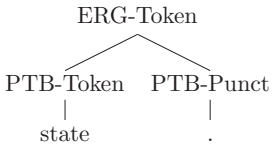

Figure 4.6: The ERG token is split into two corresponding PTB-tokens.

MWEs For multiword expressions we tokenize each single word item of the MWE, as demonstrated in Figure 4.7. In this example we see that the MWE *In fact* and the trailing comma is split into three tokens. After this PTB-like tokenization, the TnT tagger assigns each token a PTB tag. We hence frequently have a span of multiple tokens that correspond to one single ERG token. This is illustrated in Table 4.1, where *ERG token number* corresponds to the token number in the ERG-tokenized sentence.

The tokenization in Table 4.1 does not correspond directly to PTB. Specifically, we see that *well-defined* and *end-state* are split into three to-

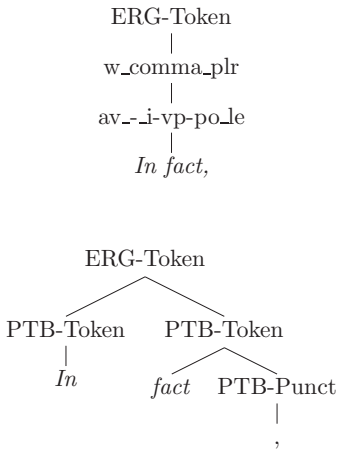

Figure 4.7: Multiword expressions extracted to single tokens prior to POS tagging.

Word token	POS Tag	ERG token number
In	IN	1
fact	NN	1
,	,	1
a	DT	2
well	RB	3
-	:	3
defined	VBN	4
end	NN	5
-	:	5
state	NN	6
.	.	6

Table 4.1: Tokenized and POS-tagged input sentence.

kens and assigned individual POS tags, whereas conventionally they would be treated as single tokens in PTB. This design choice was made to account for ERG's tokenization of hyphenated compounds, which would read this as two tokens: (*well-*, *defined*).

From a tagger performance perspective, splitting this into three tokens is suboptimal, because the TnT Tagger has been trained to tag hyphenated compounds as single tokens. However, for the supertagger the POS tag is by itself only a feature in the feature model, and there was no evidence that

a tokenization that was inconsistent with PTB was impairing the overall performance of the supertagger. On the contrary, assigning separate POS tags following the ERG tokenization proved beneficial for the supertagger in our evaluations, and this strategy was therefore preferred.

By having multiple POS-tagged tokens that map into one ERG token, one could imagine various strategies for using the supertagger input. This will be discussed in Section 4.3.2 below.

4.3.2 Supertagger Input

The supertaggers we evaluate (see Section 4.3.3) rely on a training input buffer which is a list of triples ⟨Word token, POS tag, Supertag⟩, where the supertag corresponds to the lexical type as defined in Section 4.1.3. However, unknown words in the treebanks and annotated corpus are tagged with generic lexical types. As it would be cumbersome and unnecessary having to learn which words are known and unknown to ERG, we have opted to convert these generic types back to their corresponding native lexical types, and this mapping is presented and discussed in Appendix A.

Dridan (2009) opted for a PTB-based input for the supertagger. As we can see in Figure 4.2, punctuation was assigned a simple punctuation tag, and each PTB token was assigned one ERG supertag. In the MWE *In fact* in Table 4.2, we see that both word items are assigned the same ERG supertag.

Word token	POS Tag	ERG Lexical Type
In	IN	av_-_i-vp-po_le
fact	NN	av_-_i-vp-po_le
,	,	PUNCT
a	DT	d_-_sg-nmd_le
well-defined	JJ	v_np_le
end-state	JJ	n_pp_mc-of_le
.	.	PUNCT

Table 4.2: Input buffer for the supertagger for the sentence *In fact, a well-defined end-state.* used by Dridan (2009), using lexical types as supertags.

There is an immediate problem with this approach, namely that the PTB tokenization and the ERG tokenization frequently do not correspond. When comparing the tokenization in Table 4.2 with the actual ERG tokenization as expressed in Figure 4.1, we find examples of multiple PTB-tokens that span over one single ERG lexical type (e.g. *In fact*), and vice versa (e.g. *well-defined*, where one POS tag spans over two ERG tokens). Hence, we see that the ERG lexical types for *well-* and *end-* in Table 4.2 are suppressed,

because the PTB tokens *well-defined* and *end-state* have been assigned the ERG lexical type of the final ERG token in the hyphenated compound.

In our project we opted for a training input buffer for the supertagger where each ERG token corresponds to one single ⟨Word token, POS tag, Supertag⟩ triple. Our input buffer for the sentence *In fact, a well-defined end-state.* is presented in Table 4.3, where the POS tags are derived from the TnT tags presented in Table 4.1. For MWEs and other spans where multiple POS tags are assigned for one single ERG token, the token is assigned the final non-punctuation POS-tag from the span of tokens. This heuristic assumes that most MWEs are head-final, but this simplification will however mean that some proportion of the MWEs are not handled in a manner consistent with linguistic intuition – e.g. we see that the MWE *In fact* in Table 4.3 is tagged as a noun, and not an adverb.

Word token	POS Tag	ERG Lexical Type
In fact,	NN	av_-_i-vp-po_le
a	DT	d_-_sg-nmd_le
well-	RB	av_-_dg-v_le
defined	VBD	v_np_le
end-	NN	n_-_c-ed-ns_le
state.	NN	n_pp_mc-of_le

Table 4.3: Input buffer for the supertagger for the sentence *In fact, a well-defined end-state.* used in this project.

4.3.3 Taggers

Dridan (2009) evaluated two statistical taggers in her project: TnT POS tagger (Brants, 2000) and the C&C supertagger (Clark and Curran, 2007). The taggers were trained and tested on sentences derived from the treebanks released with the 0902 version of ERG, the total size of the training set being 157,920 tokens. Almost 94% of this data was derived from the *LOGON* domain (the *jhpstg* corpus, see Table 3.1). Dridan (2009) presented a comprehensive evaluation of single tag and multiple tag accuracies for the different tagsets (as we presented in Section 4.1.4) trained on the different supertaggers. For our purpose, the single tag accuracy for lexical types is of particular interest, since we will only evaluate this tagset in the further development of our supertagger. As seen in Table 4.4, the best-performing model achieved a single tag accuracy of in the range of 0.92-0.93 for the in-domain $jhpstg_t$ data set. We note that the *ws02* and *cb* data sets, unsurprisingly, obtain a much lower accuracy. Not only were these data sets out-of-domain for the taggers used in Dridan (2009), but we also observe that their baseline accuracies are much lower, suggesting that there are in-

herent properties with these data sets that make tagging more challenging. The baseline tagger is defined as: "a simple unigram tagger that assigns the tag most frequently seen with a word, if that word was seen in the training data, or else the most frequent tag given the POS tag assigned by TnT." (Dridan, 2009, p. 70)

Data Set	Domain	Baseline	TnT	C&C
$jhpstg_t$	In-domain	0.8306	0.9287	0.9231
ws02	Out of domain	0.6594	0.8641	0.8299
cb	Out of domain	0.6752	0.8437	0.8370

Table 4.4: The best-performing model's single tag accuracy for lexical types for the in-domain $jhpstg_t$, and the out of domain *ws02* and *cb* data sets. (Dridan, 2009, p. 102).

Dridan (2009) concluded that the TnT tagger was more accurate than the C&C tagger when trained on the limited training material that was available in 2009 (see Table 4.4). However, the learning curve for the TnT tagger seemed to flatten out at around 150,000 tokens, whereas the learning curve for C&C was still rising. These findings motivated further research on the C&C tagger, especially since the amount of available training data has increased substantially over the course of this project. Particularly with the release of WikiWoods (see Section 3.4.3) the amount of automatically annotated data has surged. Given the findings in Dridan (2009), applying automatically parsed data in training seems promising:

> For both the $jhpstg_t$ and *cb* data sets, adding large amounts of automatically parsed data to the training data had the best effect on accuracy. The results on *ws02* show that adding gold standard in-domain data is still more effective, since 30,000 tokens of additional Wikipedia data gives better results that almost 700,000 tokens of parsed data. (Dridan, 2009, p. 102)

Motivated by the previous findings, we decided to evaluate the C&C tagger on additional amounts of data. Further, we used the SVM^{hmm} tagger (Joachims et al., 2009), which is an implementation of structural SVMs for sequence tagging which learns a hidden Markov model. SVM^{hmm} has a rich variety of parameter settings, and it also requires that the user defines and extracts the features. We therefore designed a simple feature model, where 14 features are extracted for each token that is tagged. Compared to the relatively fixed (and optimized) C&C supertagger, SVM^{hmm} allows for more flexibility by the user. For SVM^{hmm} we achieved the best results by using default parameter settings, and fixing the SVM trade-off slack vs. magnitude of the weight-vector (C) to equal the number of sentences that is used in the

training data. This value should increase with the size of the training data, because C is divided by the number of training examples.[3] Further, we applied a feature frequency cutoff on 2, i.e. removing all singleton features.

4.3.4 Tagging Models

For the C&C tagger, any supertag in the tagset is provided a probability $p(y \mid x)$ where y is a lexical category and x is the context:

> The context is a 5-word window surrounding the target word. Features are defined for each word in the window and for the POS tag of each word. (Clark and Curran, 2004, p. 283)

SVM^{hmm} requires that the user provides training data formatted as feature vectors for each training instance, similarly to the conventional SVM^{light} input file format (Joachims, 1999). In addition to contexts of POS tags and words, we have also defined features for the target word suffix, and features that strip the word for punctuation. Hence, the SVM^{hmm} model allows a generalization over the ERG tokens *house!* and *house*, whereas the C&C tagger will treat these tokens as two different words.

4.4 Single Tag Results

We evaluated SVM^{hmm} and the C&C tagger with an increasing amount of training data (the training data was presented in Section 3.5). We compiled the training data from WeScience data, Redwoods data (which includes WeScience) and WikiWoods data. Our aim was to measure the performance of the taggers with respect to the amount of training data and the domain from which it was derived.

As we see from Figure 4.8 to 4.10, the SVM^{hmm} and the C&C tagger have a different learning curve with respect to accuracy. Specifically, SVM^{hmm} performs better on limited training data, but it is much more expensive to train (see Figure 4.11). After 500 hours we terminated the training for both taggers. By then, the largest model trained by SVM^{hmm} consisted of 250,000 sentences. Scaling down the complexity of the feature model for the SVM^{hmm} tagger would speed up the training,[4] allowing for the use of more training data. However, these attempts consistently yielded a drop in accuracy, even though it allowed for more training instances.

We see that SVM^{hmm} consistently outperforms the C&C tagger for the amount of data that SVM^{hmm} was able to make use of. One reason for this

[3] See `http://www.cs.cornell.edu/people/tj/svm_light/svm_hmm.html` for further explanation on the parameter settings of SVM^{hmm}.

[4] The simplest way to achieve this would be to raise the feature frequency threshold that eliminates features that occur infrequently.

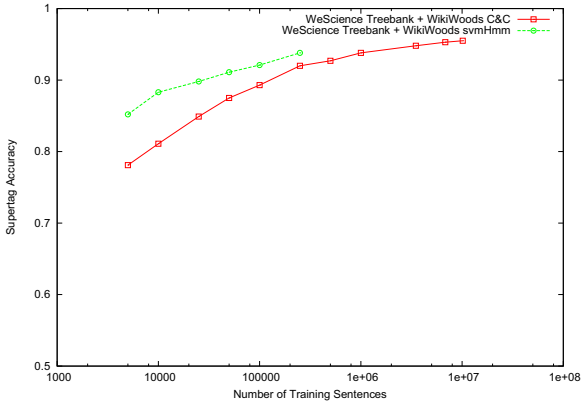

Figure 4.8: Lexical Type accuracy for C&C tagger and SVM^{hmm} on *ws12* trained on the WeScience Treebank (section 1-11, 7,636 sentences), and augmented with sentences from WikiWoods (section 2000-4000). Note that the x-axis grows logarithmically.

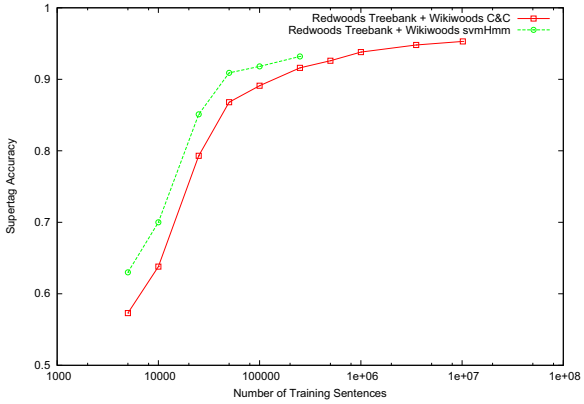

Figure 4.9: Lexical Type accuracy for the C&C tagger and SVM^{hmm} on *ws12* trained on the Redwoods Treebank (29,817 sentences), and augmented with sentences from WikiWoods (section 2000-4000).

may be that the feature model for SVM^{hmm} was targeted specifically towards ERG tokenized input, as we presented in Section 4.3.4. Hence we were able to address issues such as punctuation affixes, and capture generalization over

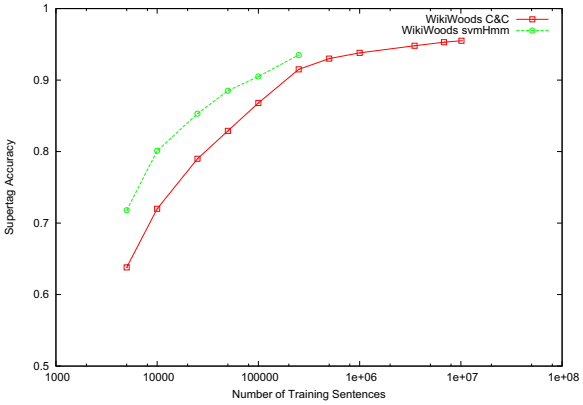

Figure 4.10: Lexical Type accuracy for the C&C tagger and SVM^{hmm} on *ws12* trained on sentences from WikiWoods (section 2000-4000).

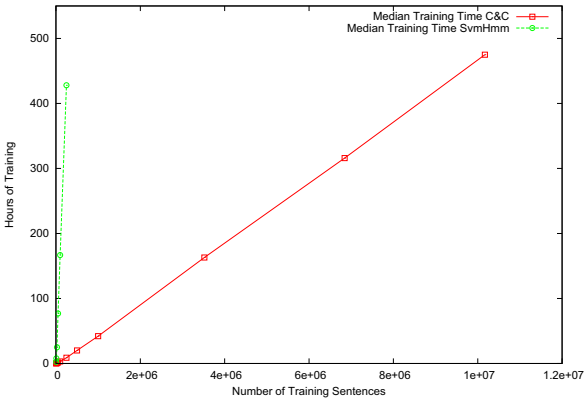

Figure 4.11: Training times for the C&C tagger and SVM^{hmm} as the amount of data increases.

tokens although the affixes differ, whereas the C&C tagger would regard each token with a different punctuation affix as a unique word. For the C&C tagger, this could lead to a substantial inflation of the lexicon compared to the SVM^{hmm} tagger, since the C&C tagger provides no generalization over word forms. For limited amounts of training data, this would also lead to a sparsity problem, where we would have many tokens in the lexicon

with relatively few occurrences. For the aforementioned reasons it thus seems likely that SVM^{hmm} is better equipped to utilize a limited amount of training data.

The training time, on the other hand, is much higher for SVM^{hmm} than for the C&C tagger. As we see in Figure 4.11, C&C scales linearly in time to the amount of input sentences, and has a steady increase in accuracy up to about 10 million input sentences (or about 150 million tokens). It seems plausible that with this massive amount of training data, the sparsity issue of a very large dictionary will no longer be of significance, and one could even imagine that distributional differences of tokens with and without punctuation affixes would benefit the feature model of the C&C tagger, as the properties of a token with and without punctuation affixes are in fact different (which is ultimately the intuition in ERG for treating punctuations similarly to inflections).

Both the C&C tagger and SVM^{hmm} were trained as a single threaded process, as neither of the taggers currently support parallelization. However, there has been successful attempts to integrate MPI support to the C&C tagger to enable parallel processing (Kummerfeld et al., 2009; Kummerfeld, 2009). Although Kummerfeld et al. (2009) are able to process about 47 million sentences, almost five times the amount of training data compared to our data set, it is however hard to see that a further increase in training data would benefit our model substantially, since we see only marginal increases in accuracy after 7 million sentences of training data (this will be further discussed in the conclusion in Section 4.5).

4.4.1 Training Data Differences

In Figure 4.8 to 4.10 we observe that the WeScience Treebank provides a better source of training data when the data set is limited, and this is particularly apparent in the range from 5,000-10,000 training derivations. However, since we only have 7,636 sentences of WeScience training derivations, this effect can only be measured on a very limited amount of data, and as we augment the training data with WikiWoods annotations, the differences are watered down until they ultimately diminish from approximately 250,000 training derivations and onwards. This effect becomes clear in Figure 4.12 and Figure 4.13. Here we find the general pattern to be that when the amount of training data is the lower range, WikiWoods and the out-of-domain Redwoods Treebank provide substantially worse results compared to a model trained on the WeScience Treebank.

In Figure 4.12 and Figure 4.13 it is also interesting to note that the model trained with Redwoods treebank performs worse that the other models, especially the WeScience model. The same tendencies can be read out of the performance of the SVM^{hmm} tagger when trained on the same data (see Figure 4.13).

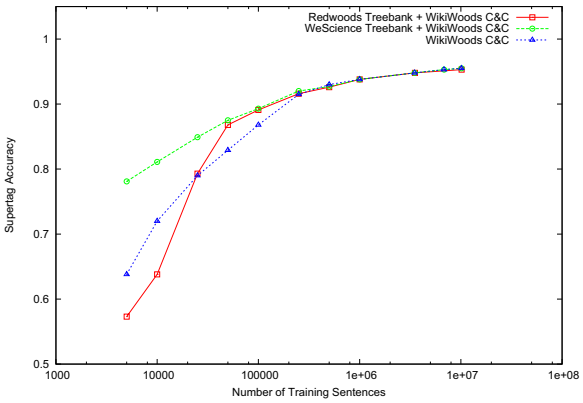

Figure 4.12: Lexical Type accuracy on *ws12* for the C&C tagger models.

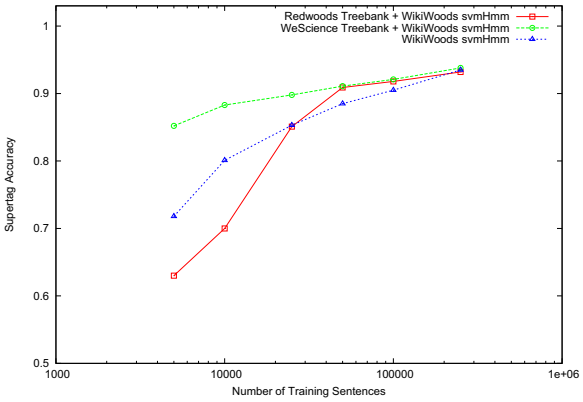

Figure 4.13: Lexical Type accuracy on *ws12* for the SVM^{hmm} tagger models.

It could be tempting to draw a conclusion that the WikiWoods and WeScience models outperform the Redwoods treebank model because they are in-domain. However, similar patterns can be read out of the out-of-domain data, albeit not to the same extent (this will be discussed in the following section). Hence, we believe there are inherent qualities with some of the corpora in the Redwoods treebank that render them less suitable as training data for the supertagger. From the data set we presented in Section 3.5, we remember that the Redwoods treebank includes the *Verbmobil* corpus, which

has a very low average sentence length, and the *jhpstg* corpus, which has a high proportion of non-English words (typically Norwegian place names). Such qualities in the training data may have affected the supertagger's ability to train an accurate model, but we have not conducted a further study into these matters.

4.4.2 Out-of-Domain Results

We have used the *cb* data set to perform a final out-of-domain evaluation of the supertaggers we developed for SVM^{hmm} and the C&C tagger. Compared to the findings on the in-domain development data set (*ws12*), we see that the performance differences between the alternative training models are much less pronounced (see Figure 4.14 and Figure 4.15), but it is still interesting to observe that even on out-of-domain data the Redwoods Treebank model performs worse that the other models when the amount of training data is small.

Figure 4.14 and Figure 4.15 show that the *cb* data set receives lower accuracy compared to *ws12*, which is consistent with the findings in Dridan (2009). The accuracy curves for variable amounts of training data are very similar to the findings for the *ws12* data set, and we see that from 100,000 training sentences onwards these differences diminish to a negligible level.

4.5 Single Tag Conclusion

Although we were not able to train SVM^{hmm} models with more than 250,000 training derivations, we successfully trained C&C tagger models with approximately 10 million training sentences. We saw a steady improvement in accuracy when more training data was added. However, from 1,000,000 training derivations onwards this increase is much less pronounced (see Figure 4.16) – by increasing the size of the training data by the order of one magnitude, the accuracy for the WeScience+WikiWoods model and the WikiWoods models go from 0.938 to 0.955 for the development *ws12* data set. The RedWoods+WikiWoods model had a marginally lower accuracy with a similar amount of training data.

Table 4.5 presents the single tag accuracy on WeScience 13 for the WikiWoods model trained on 10 million automatically annotated derivations. The *Tag Accuracy* refers to the proportion of correctly annotated lexical types, and the *Sentence Accuracy* refers to the proportion of correctly annotated sentences, and *Msec/Token* refers to the average processing time per token, excluding initialization of the tagger. We will use this model for the remainder of this thesis, and we will refer to these annotations when we use tagged annotations of *ws13* and *cb*. We observe that there are no significant difference between the tag accuracy of the test data set *ws13* (0.954) and the development data set *ws12* (0.955). This is to be expected, since we have

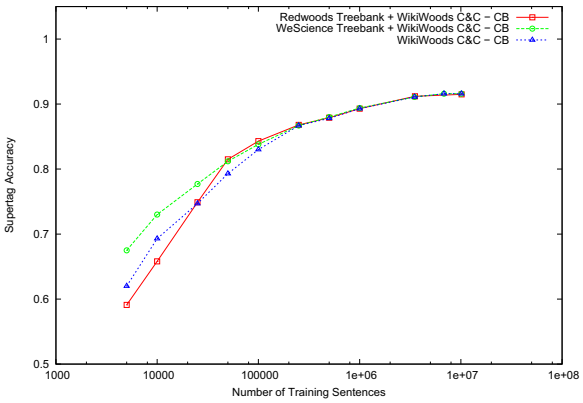

Figure 4.14: Lexical Type accuracy on *cb* for the C&C tagger models.

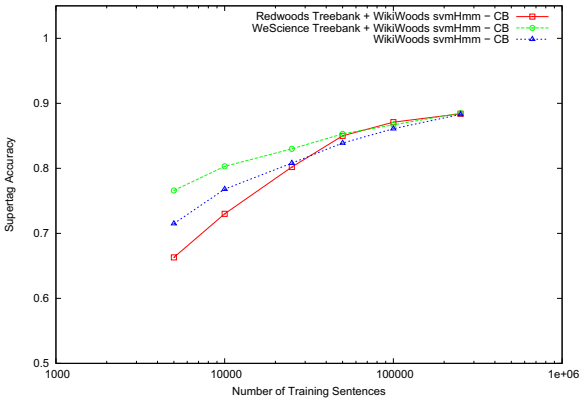

Figure 4.15: Lexical Type accuracy on *cb* for the SVM^{hmm} tagger models.

trained the C&C tagger with an out-of-the-box configuration with default parameters.

Table 4.6 presents the accuracies for the out-of-domain data. We see a distinct drop in accuracy compared to the in-domain *ws13* data set. The figures for the *jhpstg* data are comparable to the accuracies of the in-domain training models presented in Dridan (2009, p. 102), Table 4.4. These findings suggest that a model trained on a very large amount of training data can achieve comparable accuracy to an in-domain model – however, since

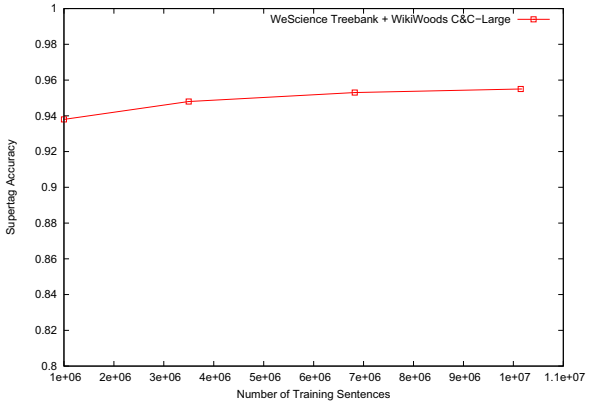

Figure 4.16: Lexical Type accuracy on *ws12* when the size of the training data is increased from 1 million to 10 million training derivations.

Test Set	***ws13***
Tag Accuracy	0.954
Sentence Accuracy	0.637
Msec/Token	0.05
# Sent in Train data	10,294,207
# Token in Train data	141,893,437

Table 4.5: Lexical Type accuracy for the in-domain *ws13*

Test Set	***cb***	***jhpstg***
Tag Accuracy	0.916	0.927
Sentence Accuracy	0.340	0.493
Msec/Token	0.05	0.06

Table 4.6: Lexical Type accuracy for the out-of-domain *cb* and *jhpstg* data sets, using the same training model as in Table 4.5.

the tokenization regime differs for the two supertaggers in question a head-to-head comparison is not warranted.

The tagging times are approximately 0.05 milliseconds per token for all test configurations for the C&C tagger.

4.6 Multiple Tags

If the object of the supertagger is to maximally reduce the search space of a subsequent parsing stage, it should ideally assign single tags of the finest possible granularity, loading off as much of the burden of the parser as it can. However, for many tasks the single tag accuracy of a supertagger is unsatisfactory, as one single mistake can lead to an unparsable sentence. A frequently used strategy is to allow a variable number of supertag candidates to be assigned to each token. In cases of uncertainty, this allows us to leave the decision to the next processing level, trading a larger search-space for a reduced risk of propagating inaccuracies to a latter parsing stage. In parsing systems that rely on supertagged input data, this strategy has been pursued by a number of research efforts. (Zhang and Clark, 2011a; Dridan, 2009; Matsuzaki et al., 2007; Clark and Curran, 2007; Ninomiya et al., 2006; Clark and Curran, 2004, inter alios) This will be further discussed in Section 5.2.1.

For determining the number of supertags that should be assigned per token, Clark and Curran (2004) suggest a probability threshold where all tags with a probability above the threshold are included:

$$\frac{p1}{\beta}$$

$p1$ corresponds to the highest tag probability and β is a user-defined value. By adjusting β we can indirectly control the number of tags that will be assigned to each token.

Dridan (2009) evaluated both single tag accuracy and multiple tag recall for the TnT and C&C supertaggers in a number of configurations. For lexical types, the recall of both the TnT and C&C taggers rise substantially when multiple tags are assigned. When assigning on average two tags per token to the $jhpstg_t$ (Tourism Data) data set, the recall for TnT goes from 0.93 to 0.98, whereas the C&C tagger goes from 0.93 to 0.96. When more tags per tokens are assigned, the C&C tagger approaches and even surpasses the recall of the TnT tagger.

4.6.1 Multitag Results

For this evaluation we apply the same training model (the WikiWoods model) and test data (*ws13*) that we used for the single tag supertagging in the last section. By adjusting the β value, we can see how the recall increases when more tags are allowed. The average tag number is the sum of tags assigned to a sentence, divided by the sentence length.

Even when keeping a relatively high β value, hence allowing only a moderate number of multitags, we see a sharp increase in tag recall. In Table 4.7 we see that assigning an average of 1.04 tags per token results in an error

β	Avg Number Tag/Token	Tag Recall	Sent Recall
0.5	1.04	0.968	0.716
0.1	1.22	0.988	0.870
0.01	1.94	0.998	0.978
0.001	3.56	0.999	0.997

Table 4.7: Multitag recall for *ws12* in various β configurations.

β	Avg Number Tag/Token	Tag Recall	Sent Recall
0.5	1.07	0.940	0.450
0.1	1.34	0.976	0.699
0.01	2.26	0.995	0.921
0.001	4.06	0.999	0.981

Table 4.8: Multitag recall for *cb* in various β configurations.

reduction of 34 % compared to the single tag accuracy. This is above all an indication that the probabilistic model of C&C is very capable of predicting its own uncertainties, and accurately predicts candidate tags in such cases. This pattern is confirmed when the β value is lowered, allowing more tags to be assigned.

As a preprocessing stage for a parser, a multi-supertagger can narrow the search-space dramatically and still maintain a very high sentence recall. As a means for lexical restriction this could greatly increase the efficiency of a chart parser by reducing the number of preterminals in the chart. Dridan (2009, p. 146) conclude that "[s]peed increases of up to 50% were achieved with no decrease in coverage or accuracy" when evaluated on the PET HPSG Parser. However, for (near-)deterministic parsers which aims to explore only a very marginal part of the search space, a multi-supertagged input buffer will increase the search-space dramatically compared to a single supertagged input buffer, and this may in its turn have a negative effect on the overall parsing performance. This will be elaborated in Section 5.2.1.

4.7 Conclusion

In this chapter we have evaluated supertagging of lexical types in ERG. Compared to prior research (in particular Dridan, 2009), where only a limited amount of training data was available, we had the entire WikiWoods (47 Million utterances) at our disposal. Given that the findings in Dridan (2009) indicate that automatically annotated data indeed contributes to boost the accuracy of the supertagger, this new source of training data did by itself call for further research on supertagging within ERG.

Our results indicate that, consistent with the findings in Dridan (2009), we can boost the accuracy of our supertaggers considerably by adding automatically annotated data to our training set. We saw the largest performance increase for the out-of-domain test set *Cathedral and the Bazaar*. Dridan (2009) reported a single tag accuracy of 0.844 for TnT, and 0.837 for C&C for the *cb* data set, whereas our best performing model scored 0.916. These figures are however not directly comparable, since the supertaggers in Dridan (2009) apply a diverging tokenization regime to ours.

With a limited amount of training data, SVM^{hmm} clearly outperformed C&C – ultimately the C&C supertagger required four times the amount of training data in order to reach the same level of accuracy that SVM^{hmm} reached with 250,000 training sentences. However, the C&C supertagger utilized a training data set which was 40 times larger than what we were able to use for the SVM^{hmm} tagger, and due to this scalability the best performing tagger models of the C&C tagger proved superior to the best performing models trained by SVM^{hmm}.

Chapter 5

Deterministic HPSG Parsing

I see only one move ahead, but it is always the correct one.
(Jose Raul Capablanca, World Chess Champion 1921-1927)

This chapter presents the architecture of the deterministic HPSG Parser which we have developed in this project. This parser, dubbed CuteForce, is a transition-based HPSG parser with a discriminative model that selects each parser action and produces analyses which, exactly or approximately conform to the English Resource Grammar (ERG, presented in Section 3.2).

The chapter is divided into two larger parts, and one initial introduction. In the first section (Section 5.1) of this chapter we will specify the expectations and requirements for a deterministic transition-based parser for an HPSG, and in particular a DELPH-IN grammar.

Section 5.2 will present the transition-based algorithm that produces the HPSG derivations. Here we will provide a thorough presentation of the data structure and the transition system of the CuteForce Parser. This section will also provide a formalization of the algorithm for the deterministic configuration, which will be applied and expanded in the remainder of this thesis.

In Section 5.3 we will present the machine learning model that navigates the search space. At each *parser configuration* (the parser configuration describes the inherent data structure of the parser, presented in Section 5.2.2) there are typically multiple search paths that can be pursued through the execution of alternative transitions. For any parser configuration that allows more than one transition, an *oracle* is used to select among the multiple alternative transitions that could be executed. This chapter will present the default configuration of CuteForce, optimized for deterministic parsing (this was also presented in Ytrestøl, 2011a). An alternative near-deterministic approach is presented in Chapter 6.

5.1 Deterministic HPSG Processing of a DELPH-IN Grammar

In Chapter 2 we defined *deterministic parsing* to be a parsing algorithm that "aims to derive a single analysis given an input string" without exploring any subanalyses that will not be part of the final derivation. From an engineering point of view, we need a framework for producing HPSG analyses that conforms to the DELPH-IN and ERG representation (more on this in Chapter 3). The prototypical representation of an HPSG derivation is a set of attribute–value pairs – the *typed feature structure* (Carpenter, 1992). However, the ERG provides a tree derivation as an additional, equivalent representation. ERG had only unary and binary productions up until version 1010. For the latest release (version 1111 was released November 2011), ERG also facilitates productions of higher rule arities. As we have customized the parser for ERG 1010, the current parsing algorithm only supports unary and binary productions. Support of higher arities would require some implementional changes to the parsing algorithm, and this is discussed in Section C.2.

The parser applies a transition-based algorithm where a multiclass classifier (the *Oracle*, presented in Section 5.3) is used to select each parser action. Although the grammatical frameworks differ, the technical ramifications for the design of such a parser are similar to the parsing approaches presented in e.g. Zhang and Clark (2011a) (CCG), Ninomiya et al. (2011) (HPSG), Nivre et al. (2007) (Dependency Grammar) and Sagae and Lavie (2005) (CFG).

Consistent with most of the deterministic parsers presented in Section 2.7, CuteForce is a transition-based parser. A transition-based algorithm is attractive to deterministic and near-deterministic parsers for at least three reasons:

1. In a deterministic configuration, a transition-based algorithm reduces the parsing to a sequence of classification choices. The length of this sequence will be linearly bounded by the input length.
2. A transition-based algorithm facilitates the use of globally defined feature function, which, as we argued Section 2.6 provides " a layer of granularity that is not easily available in a conventional packed parse forest".
3. A transition-based algorithm reduces the parsing to a tree-structured search problem (*tree-structured search problems* were introduced in Section 2.6.1).

(3) is important mainly for near-deterministic parsing approaches, which we will return to in Chapter 6.

5.2 The CuteForce Architecture

The CuteForce parser is a transition-based parser with a preprocessing stage that generates the required input buffer. In the following subsections we will present the data structure and the transition-based parsing algorithm that is applied by the parser.

5.2.1 Input Buffer

As we discussed in Chapter 4, CuteForce requires an input buffer consisting of a tokenized input sentence, equipped with part-of-speech tags and supertags. Hence, the utterence *In fact, a well-defined end-state.* will be converted to a list of triples, and each triple will provide the word form, POS tag and supertag, as presented in Table 5.1. This will be formalized in Section 5.2.2.

	Word token	POS Tag	ERG Lexical Type
1	In fact,	NN	av_-_i-vp-po_le
2	a	DT	d_-_sg-nmd_le
3	well-	RB	av_-_dg-v_le
4	defined	VBD	v_np_le
5	end-	NN	n_-_c-ed-ns_le
6	state.	NN	n_pp_mc-of_le

Table 5.1: The six input buffer items for the sentence *In fact, a well-defined end-state.*

Our design choice was selected from a range of other alternative implementations. Although we cannot provide an exhaustive list of contrasting input buffer constructions, we will discuss two alternative configurations which we find to be of particular interest to our parsing framework.

Preprocessing and Incrementality

The parsing platform's dependency on a preprocessing stage implies that the system as a whole is not incremental in the sense that it processes the input text in a single pass. Single pass incremental parsing would entail processing of an input stream of words where tokenization and sentence boundary detection is done on the fly, parallel to the syntacto-semantic analysis being derived. In speech processing or other human-computer interactions, this strategy would be particularly favorable, as it could use (human) production time for processing. From an engineering point of view, allowing for a complete on-the-fly processing would at least not seem more complex than doing it in two passes.

None of the parsers presented in Chapter 2 support input stream processing, but have an initial sentence-level preprocessing stage, typically to tokenize and derive part-of-speech/supertags. Beuck et al. (2011a) present a comprehensive study of the cost of incremental POS tagging. Naive incremental POS tagging lacks information about the next words in the stream, because a POS tag is assigned as soon as the token is read. This strategy is contrasted to allowing a two word delay, hence have a lookahead window of two tokens on the right hand side (when parsing left to right). By using this lookahead windows, the accuracy of the tagger increased from 95.63% to 96.57%, which was only marginally lower than the non-incremental result (96.64%). Further, in Beuck et al. (2011b) the same authors adapt pre-existing dependency parsers (MaltParser, Nivre et al., 2007 and WCDG, Schrøder, 2002) to achieve fully incremental processing, amongst others by taking advantage of the incremental POS tagger introduced in Beuck et al. (2011a).

The findings in Beuck et al. (2011a) and Beuck et al. (2011b) suggest that applying a similar approach would be a feasible strategy for our platform if we would enforce single pass incremental processing for the parsing system as a whole. However, this is not pursued further in this project, and we will assume a preprocessing stage prior to the deterministic syntacto-semantic parsing.

Multitagging

When the transition-based parsing is launched, the preterminals in the derivation have already been assigned by the supertagger. In the event of an incorrectly tagged preterminal, we may have an input buffer that proves unparsable due to the incorrectly assigned lexical entry. As discussed in Section 4.6, the supertagger would gain a substantial increase in sentence recall when allowed to tag multiple tags in events of uncertainty – the sentence recall goes from 0.633 to 0.978 when on average two tags are assigned per input token (these scores are collected from Table 4.5 and 4.7).

The leap in recall suggests that multi-supertagging is an efficient way of reducing the number of preterminal candidates while retaining a high sentence recall. For chart parsers this could be particularly useful, as it would dramatically reduce the search space, yet run a very low risk of pruning away the correct preterminal (Dridan, 2009; Clark and Curran, 2004).

The strategy is also applied by prominent transition-based parsers, such as Zhang and Clark (2011a). In the transition system of these parsers, the shift-operation selects a preterminal from a set of candidates, and pushes this candidate to the stack. Although non-deterministic parsers can benefit greatly in terms of efficiency by assigning a set of candidate supertags, the intuitions for a deterministic configuration are not as unconditional. A deterministic parser would only benefit from utilizing an input buffer

with multitags if the parser itself is better equipped to identify the correct preterminal than the supertagger was. Theoretically one could argue that this should be the case, considering that the parser can utilize syntactic and semantic information derived from the partial analysis when choosing a preterminal candidate, in addition to all the information the supertagger provided. However, for our case this would come with a cost: The machine learning model of CuteForce relies heavily on the lexical types as features. Introducing multiple tags would have blown up the complexity of the feature model, since it would have had to consider multiple lexical entries and probabilities per token.

Although it could be interesting to evaluate how CuteForce would perform if the parsing algorithm and machine learning model were adapted to process multitags in the input buffer, we are due to the aforementioned reasons skeptical that it would actually have contributed positively in a deterministic configuration. For the scope of this project, we have thus only evaluated single tagged preterminal input.

5.2.2 Data Structure for the Parser

In the previous section we introduced the input buffer. In this section we will formalize the data structure that is used by CuteForce. The sentence input buffer β is a list of triples of word forms W, part of speech-tags POS, and lexical types LT derived from a sentence:
$\beta = (W_1, POS_1, LT_1), (W_2, POS_2, LT_2), \ldots, (W_n, POS_n, LT_n)$. Given a set of HPSG rules R and a sentence buffer β, a *parser configuration* is a triple $c = (\iota, \pi, \delta)$ where:

- ι is the current input buffer position in β. Because we consider the input buffer to be immutable, ι is used as a reference for how much of the input buffer that is seen.
- π is a stack of passive edges instantiating an HPSG rule. The passive edge in our framework is a five-tuple:
  passive-edge = ⟨id, start, end, HPSG Category, Daughters<>⟩
  The passive edge used in CuteForce is similar to the concept of a passive edge in the conventional chart parsing scheme. The *id* is a unique number for each passive edge in π, *start* and *end* denote the start and end vertex for the edge, and *HPSG Category* is either a lexical type/preterminal or a lexical/syntactic rule. The list of daughters can be empty (for preterminals), or contain up to two daughters. In the data structure, the daughters are pointers to other passive edges in π. Each transition will push one new edge to π, and there is no *pop*-operation associated with this stack.
- δ is a stack of 'detached edges'. A detached edge is a passive edge in the π stack which is not an ancestor of the top element on the

π stack. This detached edge is expected to be the left daughter of a binary rule. Following a so-called *late commitment* strategy, we postpone predicting the category of the binary rule until the right-hand side daughter is found (early versus late commitment will be further discussed in Appendix C). Up until this point, the detached edges will remain 'on hold' as root nodes in a tree (possibly containing only one element) which is not an ancestor of the top element of the π stack. The *delta* stack is popped once the right daughter is found, where upon a new edge spanning the left to the right daughter is added to π. Hence, the edge pointed to by the popped element from δ is no longer detached from $\pi_{(0)}$

The stack of passive edges π makes up the full, connected HPSG representation of the input string if the string is accepted. Further, as long as $\delta = \emptyset$, the (possibly partial) HPSG analysis on π is a fully connected tree derivation. Initially π consists of the first preterminal from the buffer β, and ι is hence initialized to 1. As ι is incremented (through a DETACH transition, see Section 5.2.3), the next buffer element from β is instantiated as a passive edge and added to π.

5.2.3 Transition System

A *transition* is a function that conducts a specific operation on the data structure of the parser. For CuteForce this implies a manipulation of its current configuration $c = (\iota, \pi, \delta)$. CuteForce applies four different transitions. All transitions manipulate the δ or the π stack, or ι. Whereas the δ stack can be both pushed and popped, the π stack only allows a push operation, and each transition will add one new passive edge to the π stack. ι can be incremented up to the point where it equals the buffer length (at this point there are no further unprocessed items in the buffer).

After a successful parse, the final analysis of a sentence which is reached after n transitions can be derived from c_n (or specifically from π in c_n), and we can formalize this transition sequence T:

$$T = \arg\max_{t \in N(c_0)} P(t \mid c_0), \arg\max_{t \in N(c_1)} P(t \mid c_1), \ldots, \arg\max_{t \in N(c_n)} P(t \mid c_n) \tag{5.1}$$

where the function $N(c_n)$ (*Next*) provides the set of applicable transitions for each parser configuration c_n. Conceptually we can consider a transition $t \in N(c_n)$ to be a pair, containing the transition category, and a corresponding (pseudo-)probability. The transition t of highest probability in $N(c_n)$, $\arg\max_{t \in N(c_n)} P(t \mid c_n)$, will be executed for each parse configuration (c_n).

When referring to the elements on the stack, we index them relative to the top element, where $\delta_{(0)}/\pi_{(0)}$ refer to the top element of the stack, $\delta_{(1)}/\pi_{(1)}$ denote the next stack element, etc.

The four transitions are:

- DETACH — This transition pushes a pointer to a 'detached edge' to the δ stack. This stack element points to the top element on the π stack ($\pi_{(0)}$), expressing the hypothesis that $\pi_{(0)}$ is the left daughter of a binary HPSG rule. This element will remain on the δ stack until the right-hand side daughter is found. Further, ι is incremented, and the passive edge $\langle \text{id}, \iota - 1, \iota, LT_{(\iota)}, \emptyset \rangle$ is pushed to π. Effectively, the passive edge which is now pointed to by the top element of the δ stack ($\delta_{(0)}$) becomes *detached*. This means that this edge, and its ancestors, are not an ancestor of $\pi_{(0)}$. This transition is prohibited if ι equals the length of β, which implies that there are no further unread elements in β.
- UNIT(R^1) — This transition pushes a new unary passive edge to π. The transition is parameterized with a unary HPSG rule (R^1), and pushes the passive edge $\langle \text{id}, \text{start}(\pi_0), \text{end}(\pi_0), R^1, < \pi_{(0)} > \rangle$ to π. The unary daughter will be a pointer to the prior top element of the π stack before the new edge is pushed to π. This transition is prohibited if nine consecutive unit transitions have been executed. This restriction is imposed to avoid the parser entering an infinite unary production loop. This can happen when the oracle (more on this in Section 5.3) predicts a reoccurring sequence of unit transitions, in which case the π stack keeps growing, but without processing new elements from the buffer. This strategy is parallel to the deterministic CFG parser developed by Sagae and Lavie (2005), where a non-unary production is forced after three consecutive unary transitions.
- PASSIVE(R^2) — This transition pops the δ stack and a new binary passive edge is pushed to π. This new passive edge spans from the start index of the popped element from δ, $start(\delta_0)$, to ι. The left daughter will be the passive edge referred to by δ_0, and the right daughter will be $\pi_{(0)}$. The transition is parameterized with a binary HPSG rule R^2. This new passive edge can be formalized as: $\langle \text{id}, \text{start}(\delta_0), \iota, R^2, < \delta_0, \pi_0 > \rangle$, where δ_0 and π_0 represent the top elements of the δ/π stack before any stacks are popped/pushed. Effectively, the passive edge pointed to by δ_0 which was previously detached, is now attached as the left daughter of the new top element of π. This transition is prohibited if $\delta = \emptyset$.
- ACCEPT(S) — This transition adds a start symbol (S) to the HPSG derivation, and terminates the parse. The stack operation is identical to a UNIT(R^1) transition, except that a unary rule is selected among the start symbols, whereupon the stack π, which contains the final parse derivation for the input sentence β, is returned. This transition is only valid if (1) ι equals the input buffer length and (2) the δ stack is empty, ensuring that the π stack holds a fully connected HPSG

derivation.

For the current transition system and data structure (presented above), Shift-Reduce parser would have sufficed to facilitate the parsing operations we have introduced for CuteForce. In Appendix C we discuss the similarities of the current data structure and transition system of CuteForce, and motivate why we have opted not to define CuteForce within a Shift-Reduce scheme.

Complexity This transition-system employed in a deterministic configuration ensures a linear time-complexity for the parser. The maximum number of transitions for a utterance of length n is: $10(2n-1)$. This equation can be reached because the number of DETACH and PASSIVE transitions equals $n-1$, there is one ACCEPT transition, and there can be maximally 9 consecutive unit transitions before a DETACH, PASSIVE or ACCEPT transition must be executed. We found the observed average number of transitions executed per sentence to be less than $4n$ (in the range of 3.8 and 4.0 transitions per input token), and this average was consistent both for the development data set (*ws12*), the test sets (*ws13* and *cb*), and when parsing large amounts of WikiWoods data (500,000 sentences from Section 4000 onwards).[1] We will exemplify the parser actions in the following section.

5.2.4 Parsing Example

In this section we will go through the parsing stages that are required to generate the HPSG analysis in Figure 5.1. We observe that the input buffer consists of six elements (see Table 5.2) – these elements constitute the terminals and preterminals in the tree derivation. The POS tags are not used in the HPSG analysis, but are utilized by the oracle.

	Word token	POS Tag	ERG Lexical Type
1	This	DT	n_-_pr-dei-sg_le
2	is	VBZ	v_prd_is_le
3	sometimes	RB	av_-_i-vp-j_le
4	called	VBN	v_np-np_only_le
5	a	DT	d_-_sg-nmd_le
6	“loop.”	NN	n_-_c_le

Table 5.2: Input buffer for the sentence *This is sometimes called a ”loop.“*

Tables 5.3 and 5.4 present all transitions which are required in order to produce the correct HPSG derivation of the sentence *This is sometimes*

[1] The data sets were introduced and discussed in Section 3.5.

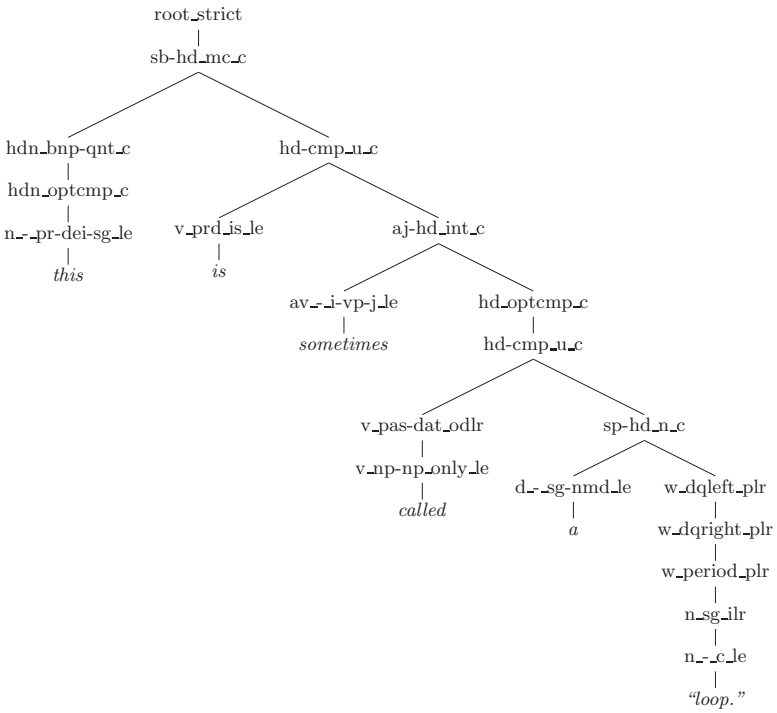

Figure 5.1: Parse tree for the derivation *This is sometimes called a "loop."*

called a "loop.", as presented in the tree derivation in Figure 5.1. It should be stressed that this is the only transition sequence that can generate this exact derivation.

The transition number is denoted in the left-most columns of table 5.3 and 5.4. The *Transition* columns are reserved for the transition categories, parameterized with an HPSG Rule name (for the UNIT and PASSIVE transition), or with a start symbol (for the ACCEPT transition). The δ column holds the stack of detached edges, which are pointers to edges on the passive edge stack (these pointers are expressed by their edge id). The ι column refers to the current buffer item which is being processed. Due to space limitations, the π columns in Table 5.3 and 5.4 are truncated, and only the top three edges on the π stack are shown. The full π stack after transition 19 (the ACCEPT(S) transition) is presented in Table 5.5.

Figures 5.2 to 5.7 represent the π stack as tree derivations. The top of the π stack, $\pi_{(0)}$, will always be the passive edge with the highest edge id. Disconnected edges are represented as trees that precede the $\pi_{(0)}$ and its

#	**Transition**	δ	ι	π
0	Initial Conf	$\emptyset$	1	$\langle 100, 0, 1$,n_-_pr-dei-sg_le,$\emptyset\rangle$
1	UNIT (hdn_optcmp_c)	$\emptyset$	1	$\langle 101, 0, 1$,hdn_optcmp_c,<100>$\rangle$ $\langle 100, 0, 1$,n_-_pr-dei-sg_le, $\emptyset\rangle$
2	UNIT (hdn_bnp-qnt_c)	$\emptyset$	1	$\langle 102, 0, 1$,hdn_bnp-qnt_c, <101>$\rangle$ $\langle 101, 0, 1$,hdn_optcmp_c, <100>$\rangle$ $\langle 100, 0, 1$,n_-_pr-dei-sg_le, $\emptyset\rangle$
3	DETACH	[102]	2	$\langle 103, 1, 2$,v_prd_is_le, $\emptyset\rangle$ $\langle 102, 0, 1$,hdn_bnp-qnt_c, <101>$\rangle$ $\langle 101, 0, 1$,hdn_optcmp_c, <100>$\rangle$ $\langle \ldots\rangle$
4	DETACH	[103, 102]	3	$\langle 104, 2, 3$,av_-_i-vp-j_le, $\emptyset\rangle$ $\langle 103, 1, 2$,v_prd_is_le, $\emptyset\rangle$ $\langle 102, 0, 1$,hdn_bnp-qnt_c, <101>$\rangle$ $\langle \ldots\rangle$
5	DETACH	[104, 103, 102]	4	$\langle 105, 3, 4$,v_np-np_only_le, $\emptyset\rangle$ $\langle 104, 2, 3$,av_-_i-vp-j_le, $\emptyset\rangle$ $\langle 103, 1, 2$,v_prd_is_le, $\emptyset\rangle$ $\langle \ldots\rangle$
6	UNIT (v_pas-dat_odlr)	[104, 103, 102]	4	$\langle 106, 3, 4$,v_pas-dat_odlr, <105>$\rangle$ $\langle 105, 3, 4$,v_np-np_only_le, $\emptyset\rangle$ $\langle 104, 2, 3$,av_-_i-vp-j_le, $\emptyset\rangle$ $\langle \ldots\rangle$
7	DETACH	[106, 104, 103 102]	5	$\langle 107, 4, 5$,d_-_sg-nmd_le, $\emptyset\rangle$ $\langle 106, 3, 4$,v_pas-dat_odlr, <105>$\rangle$ $\langle 105, 3, 4$,v_np-np_only_le, $\emptyset\rangle$ $\langle \ldots\rangle$
8	DETACH	[107, 106, 104 103, 102]	6	$\langle 108, 5, 6$,n_-_c_le, $\emptyset\rangle$ $\langle 107, 4, 5$,d_-_sg-nmd_le, $\emptyset\rangle$ $\langle 106, 3, 4$,v_pas-dat_odlr, <105>$\rangle$ $\langle \ldots\rangle$
9	UNIT (n_sg_ilr)	[107, 106, 104 103, 102]	6	$\langle 109, 5, 6$,n_sg_ilr, <108>$\rangle$ $\langle 108, 5, 6$,n_-_c_le, $\emptyset\rangle$ $\langle 107, 4, 5$,d_-_sg-nmd_le, $\emptyset\rangle$ $\langle \ldots\rangle$
10	UNIT (w_period_plr)	[107, 106, 104 103, 102]	6	$\langle 110, 5, 6$,w_period_plr, <109>$\rangle$ $\langle 109, 5, 6$,n_sg_ilr, <108>$\rangle$ $\langle 108, 5, 6$,n_-_c_le, $\emptyset\rangle$ $\langle \ldots\rangle$

Table 5.3: The first ten transitions for the parse of the sentence *This is sometimes called a "loop.“*. The remaining transitions are found in Table 5.4

.

ancestors in Figure 5.2 to 5.7. In Figure 5.3 we see that edge 107 represent $\pi_{(0)}$. The disconnected edges are represented as four disconnected trees,

#	**Transition**	δ	ι	π
11	UNIT (w_dqright_plr)	[107, 106, 104 103, 102]	6	⟨111, 5, 6,w_dqright_plr, <110>⟩ ⟨110, 5, 6,w_period_plr, <109>⟩ ⟨109, 5, 6,n_sg_ilr, <108>⟩ ⟨. . .⟩
12	UNIT (w_dqleft_plr)	[107, 106, 104 103, 102]	6	⟨112, 5, 6,w_dqleft_plr, <111>⟩ ⟨111, 5, 6,w_dqright_plr, <110>⟩ ⟨110, 5, 6,w_period_plr, <109>⟩ ⟨. . .⟩
13	PASSIVE (sp-hd_n_c)	[106, 104, 103 102]	6	⟨113, 4, 6,sp-hd_n_c, <107,112>⟩ ⟨112, 5, 6,w_dqleft_plr, <111>⟩ ⟨111, 5, 6,w_dqright_plr, <110>⟩ ⟨. . .⟩
14	PASSIVE (hd-cmp_u_c)	[104, 103, 102]	6	⟨114, 3, 6,hd-cmp_u_c, <106,113>⟩ ⟨113, 4, 6,sp-hd_n_c, <107,112>⟩ ⟨112, 5, 6,w_dqleft_plr, <111>⟩ ⟨. . .⟩
15	UNIT (hd_optcmp_c)	[104, 103, 102]	6	⟨115, 3, 6,hd_optcmp_c, <114>⟩ ⟨114, 3, 6,hd-cmp_u_c, <106,113>⟩ ⟨113, 4, 6,sp-hd_n_c, <107,112>⟩ ⟨. . .⟩
16	PASSIVE (aj-hd_int_c)	[103, 102]	6	⟨116, 2, 6,aj-hd_int_c, <104,115>⟩ ⟨115, 3, 6,hd_optcmp_c, <114>⟩ ⟨114, 3, 6,hd-cmp_u_c, <106,113>⟩ ⟨. . .⟩
17	PASSIVE (hd-cmp_u_c)	[102]	6	⟨117, 1, 6,hd-cmp_u_c, < 103, 116 >⟩ ⟨116, 2, 6,aj-hd_int_c, <104,115>⟩ ⟨115, 3, 6,hd_optcmp_c, <114>⟩ ⟨. . .⟩
18	PASSIVE (sb-hd_mc_c)	$\emptyset$	6	⟨118, 0, 6,sb-hd_mc_c, <102,117>⟩ ⟨117, 1, 6,hd-cmp_u_c, <103,116>⟩ ⟨116, 2, 6,aj-hd_int_c, <104,115>⟩ ⟨. . .⟩
19	ACCEPT (root_strict)	$\emptyset$	6	⟨119, 0, 6,root_strict, <118>⟩ ⟨118, 0, 6,sb-hd_mc_c, <102,117>⟩ ⟨117, 1, 6,hd-cmp_u_c, <103,116>⟩ ⟨. . .⟩

Table 5.4: Continue from Table 5.3: The last nine transitions for the parse of the sentence *This is sometimes called a "loop.“*.

with the edges 106, 104, 103 and 102 as the root nodes. These disconnected edges are equivalent to the detached edges in this parser configuration, and the δ stack are made up of pointers to these root nodes (see also Table 5.3, transition 7).

Passive Edges (π)
$\langle 119, 0, 6$,root_strict, <118>$\rangle$
$\langle 118, 0, 6$,sb-hd_mc_c, <102,117>$\rangle$
$\langle 117, 1, 6$,hd-cmp_u_c, <103,116>$\rangle$
$\langle 116, 2, 6$,aj-hd_int_c, <104,115>$\rangle$
$\langle 115, 3, 6$,hd_optcmp_c, <114>$\rangle$
$\langle 114, 3, 6$,hd-cmp_u_c, <106,113>$\rangle$
$\langle 113, 4, 6$,sp-hd_n_c, <107,112>$\rangle$
$\langle 112, 5, 6$,w_dqleft_plr, <111>$\rangle$
$\langle 111, 5, 6$,w_dqright_plr, <110>$\rangle$
$\langle 110, 5, 6$,w_period_plr, <109>$\rangle$
$\langle 109, 5, 6$,n_sg_ilr, <108>$\rangle$
$\langle 108, 5, 6$,n_-_c_le, $\emptyset\rangle$
$\langle 107, 4, 5$,d_-_sg-nmd_le, $\emptyset\rangle$
$\langle 106, 3, 4$,v_pas-dat_odlr, <105>$\rangle$
$\langle 105, 3, 4$,v_np-np_only_le, $\emptyset\rangle$
$\langle 104, 2, 3$,av_-_i-vp-j_le, $\emptyset\rangle$
$\langle 103, 1, 2$,v_prd_is_le, $\emptyset\rangle$
$\langle 102, 0, 1$,hdn_bnp-qnt_c, <101>$\rangle$
$\langle 101, 0, 1$,hdn_optcmp_c, <100>$\rangle$
$\langle 100, 0, 1$,n_-_pr-dei-sg_le, $\emptyset\rangle$

Table 5.5: The π stack representing the final HPSG analysis after transition 19 in Table 5.4. This stack represents the fully connected HPSG analysis for the sentence *This is sometimes called a "loop."*.

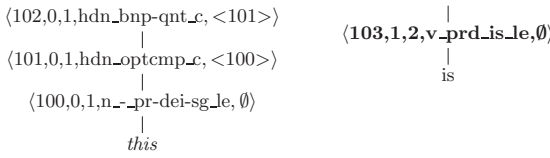

Figure 5.2: The π stack formatted as a disconnected tree derivation after 3 transitions. We see that the first four edges from Table 5.5 are represented (edges 100-103). A *detached* tree fragment, with edge 102 as the root node, is represented prior to the edge on $\pi_{(0)}$ (edge 103, in **bold**), and its possible ancestors. *Detached* in this sense means that it is not an ancestor of $\pi_{(0)}$, as we can overtly see from this representation. In the data structure, this is reflected by the δ stack containing one pointer, namely to edge *102*.

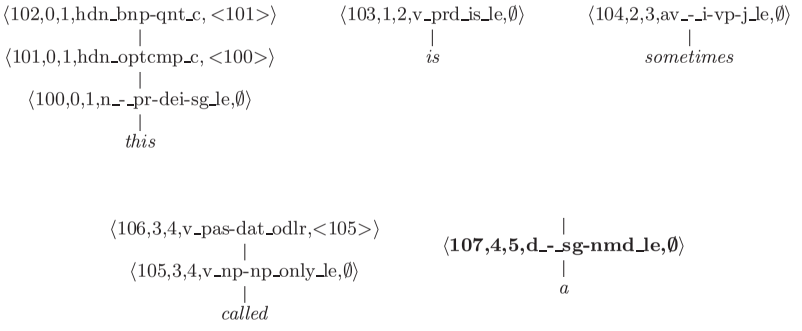

Figure 5.3: The π stack after 7 transitions. There are four detached tree fragments, reflected by the δ stack, which currently holds pointers to the edges $[106, 104, 103, 102]$. These tree fragments will remain static until a PASSIVE transition is executed. Table 5.3, transition 7, presents the same data structure as depicted here.

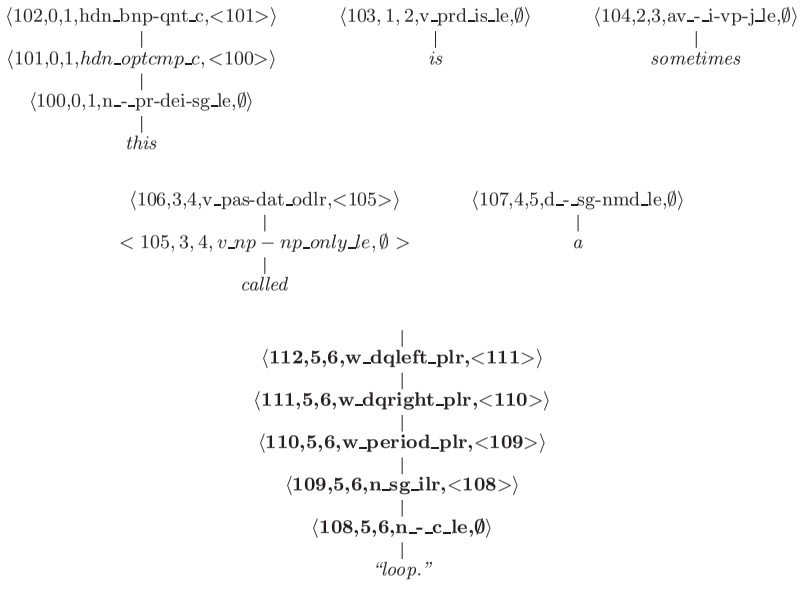

Figure 5.4: The π stack after 12 transitions. We see the five detached tree fragments, in addition to edge 112 on $\pi_{(0)}$, and its ancestors. Now ι equals the sentence length, which means that any further DETACH transitions are prohibited. The only transitions which are valid in this state are UNIT and PASSIVE (ACCEPT is prohibited as long as there are detached edges, i.e. that the δ stack is not empty).

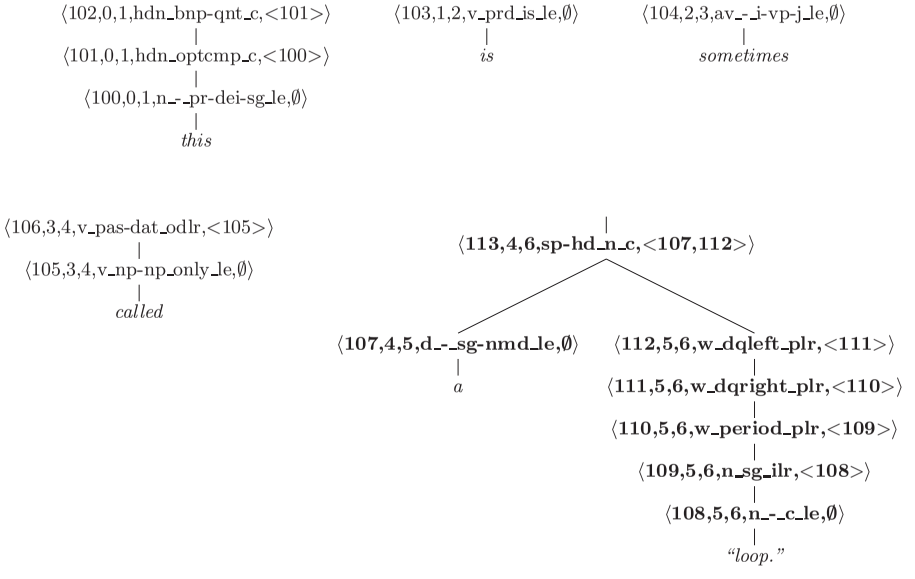

Figure 5.5: The π stack after 13 transitions. A PASSIVE(sp-hd_n_c) transition was executed, and the δ stack was popped. The element that was popped pointed to the passive edge 107, which is now attached to the top edge of the π stack. The δ stack now consists of four detached tree fragments, with the root nodes [106,104,103,102].

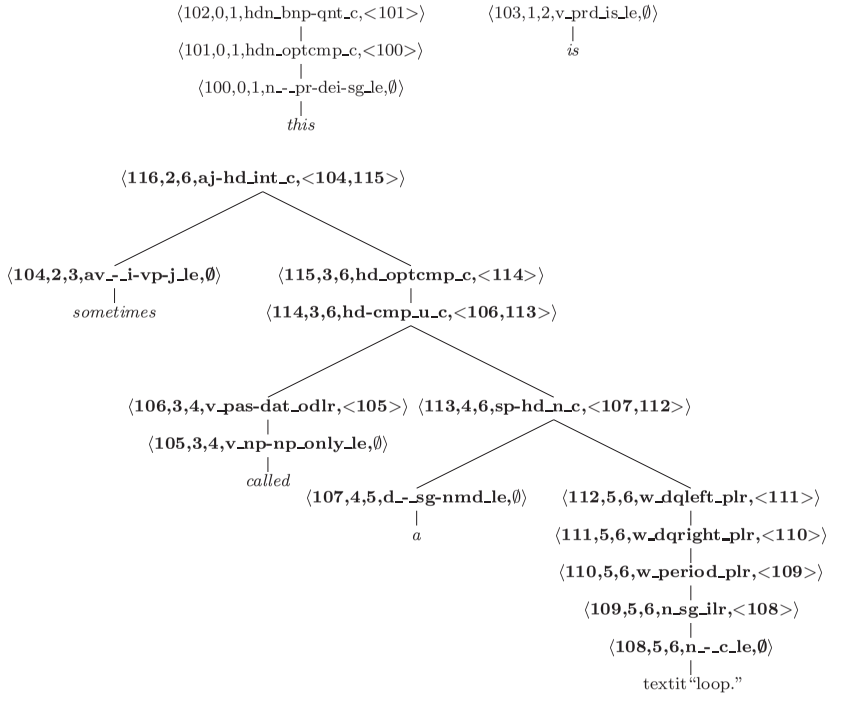

Figure 5.6: The π stack after 16 transitions. A PASSIVE(aj-hd_int_c) transition was executed, and there are currently two detached tree fragments which need to be attached through a PASSIVE transition before the ACCEPT(S) transition is valid.

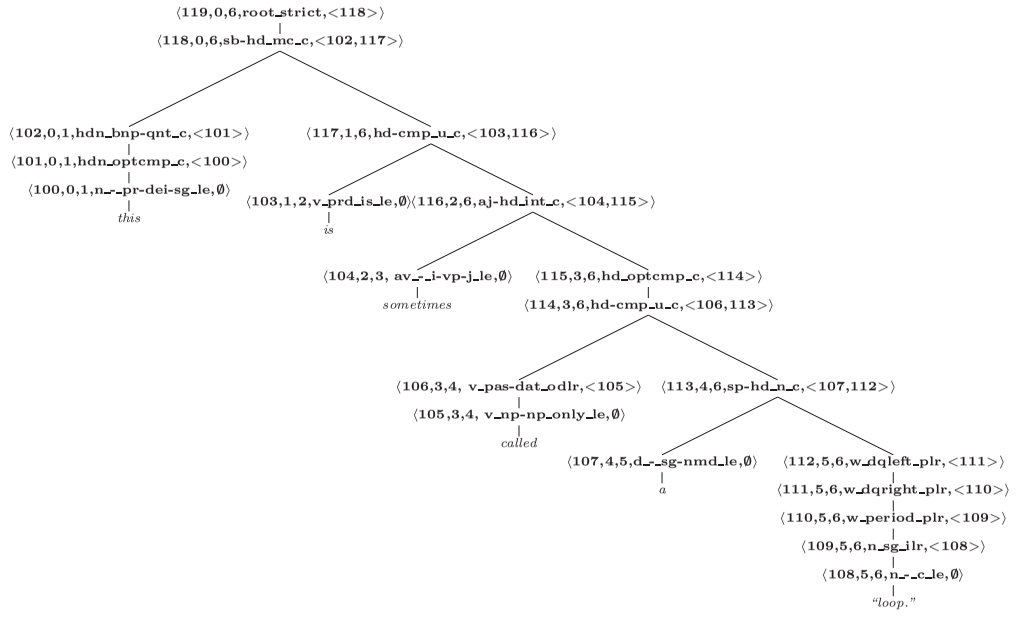

Figure 5.7: The final HPSG derivation after 19 transitions. The ACCEPT(root_strict) transition was executed, which implies that there are no detached tree fragments. The π stack of fully connected edges (also presented in Table 5.5) is easily converted into a conventional tree derivation, as seen above. Notice that this tree derivation is equivalent to the derivation is Figure 5.1, with the exception that Figure 5.1 only presents the HPSG categories or lexical types, as the rest of the edge description is superfluous when presented as a tree structure.

CuteForce more robust for experimentations with alternative parsing strategies, and still maintain an equivalent access to the features as defined in the current feature model for CuteForce.

versatile framework that does allow for customizations that goes beyond a Shift-Reduce scheme.

5.2.5 Parsing Configuration Mode

CuteForce has three different *parsing modes*. As we discussed in Section 5.2.3, for each parser configuration, the oracle (to be introduced in Section 5.3) will select the next transition from a set of *applicable* transitions. The applicability of a transition is initially determined by the transition system, and its built-in constraints (presented in Section 5.2.3). Further, the parsing modes may impose additional filtering on the transitions that are licensed by the transitions system. The three parsing modes are: HPSG unification mode, CFG approximation mode and unrestricted mode.

HPSG Unification Mode In HPSG unification mode, the parser validates that the UNIT and PASSIVE transitions imply a valid unification. This is achieved through an XML-RPC interface to the third-party LKB platform (Copestake, 2002). In this parsing mode, the parsing oracle will only select among the transitions that yield a valid unification, effectively purging the transitions that breach a grammatical constraint. If there are no valid transitions available, parsing fails for the given sentence, and no analysis is returned. All parses that terminate with the ACCEPT transition are ensured to have produced a valid HPSG derivation.

CFG Approximation Mode In CFG mode, a CFG approximation of the ERG is employed by the oracle. The CFG approximation consists of CFG rules harvested from the parser's training data, augmented with derivations from WikiWoods. After having experimented with CFG approximations derived from various amounts of annotated data, our final configuration employs a CFG approximation extracted from a total of 300,000 sentences. Each ERG rule instantiation, using the identifiers shown in Figure 5.1 as non-terminal symbols, will be treated as a CFG rule, and all transitions will be validated against the set of CFG rules. In total, there are 30,050 valid binary production rules and 6,568 unary production rules in the CFG approximation that we apply.

In this mode, the parsing oracle will only select among the transitions which yield a CFG projection that is found among the CFG rules in the CFG approximation of the ERG. If the parser arrives at a state where the CFG filter blocks all further transitions, parsing fails.

Unrestricted Mode In unrestricted mode, the oracle selects the highest ranked transition from the set of parser operations that are licensed by the transition system, without any further filtering. In this setting, the parser should reach 100% coverage.

5.3 Oracle

To navigate the search space, we apply a discriminative machine learning algorithm to predict the parser actions. We call this mechanism the *Oracle* (Kay, 2000). The parsing is hence reduced to a classification problem. As we mentioned in Section 5.2.3, each HPSG analysis is derived from one unique sequence (T) of parser actions, where the function $N()$ returns the set of the next applicable transitions, and their corresponding pseudo-probabilites and the function $P()$ retrieves the probability from the transition/probability pair returned by $N()$ given a parser configuration c_n:

$$T = \arg\max_{t \in N(c_0)} P(t \mid c_0), \arg\max_{t \in N(c_1)} P(t \mid c_1), \ldots, \arg\max_{t \in N(c_n)} P(t \mid c_n) \tag{5.2}$$

We can consider this a deterministic implementation of a *History-based model*, introduced by Black et al. (1992). The ranking of each transition in the sequence T relies on the *history*, i.e. the configurations up until the current configuration c_n.

$$\arg\max_{t \in N(c_n)} P(t \mid \Phi(c_0, c_1, \ldots, c_n)) \tag{5.3}$$

The function Φ maps the current parse configuration, or *history*, to a feature vector. The vector is further defined through a set of *feature templates*, which will be presented in Section 5.3.4. The selection of features that defines the mapping of each parse configuration to the feature vector is called the *Feature Model*, and the feature model for CuteForce will be presented in Section 5.3.6.

For training we use LibLinear (Fan et al., 2008), which provides a number of solvers. In CuteForce, we have used the implementation of SVM multiclass classification of Crammer and Singer's formula (Keerthi et al., 2008; Crammer and Singer, 2002; Vapnik, 1995), which gave better results than other learners evaluated during the development.[2]

5.3.1 Training Data

The model is trained and tested on the WeScience Treebank, a branch of the hand-annotated LinGO Redwoods treebank (Ytrestøl et al., 2009; Oepen

[2]The other LibLinear parameters in the final training cycle were set to: `-e 0.001 -B -1 -c 0.2`

et al., 2002b, see Section 3.4.2). Section 1-11 is used for training, Section 12 is used in development and Section 13 is held-out for testing. The training data is further augmented with additional sentences from other Redwoods treebanks, and derivations from the automatically annotated WikiWoods corpus. The corpora that are used in training and testing of the CuteForce parsing oracle are identical to the data used to train and test the supertagger, and are presented in Section 3.5. Chapter 3 provides a more in-depth presentation of the training data used in this thesis. Further, Section 7.3 gives an evaluation of the training data applied by CuteForce with respect to the accuracy of the Oracle.

5.3.2 Single-Stage versus Two-Stage Classification

MaltParser (Nivre et al., 2007) supports two alternative classifier setups, which we will refer to as *single-stage* and *two-stage* classification. In two-stage classification, an initial model predicts the parser operation, and secondary models predict the label if the parser operation is parameterized. Intuitively there are good reasons for doing this: By dividing the prediction of the parser action to a two-staged event, one can employ individual feature models explicitly designed to predict among a smaller group of parse labels, hence hoping to achieve more accurate predictions.

This strategy was also evaluated for CuteForce. In this two-stage configuration, an initial classifier chooses among the four transitions (UNIT, PASSIVE, DETACH and ACCEPT), and three secondary classifiers predict the parse labels, i.e. the categories for the parameterized transitions UNIT, PASSIVE and ACCEPT. Further, we evaluated single-stage classification where the oracle predicts not only the parser transition's categories, but also predicts the labels for the parameterized parsing actions (i.e. the UNIT, PASSIVE and ACCEPT transition) in one multiclass prediction. This multiclass prediction consists of 274 classes, with one DETACH transition, four ACCEPT transitions, 135 UNIT transitions and 134 PASSIVE transitions. This will be further discussed in Section 7.5, where the parsing results for the two-stage classification setup will be evaluated and compared to the single-stage classification setup.

5.3.3 Feature Functions

CuteForce is equipped with a rich feature selection optimized for a large (100,000+) set of training derivations. In our training data of 150,000 sentences, we have approximately six million training instances, where each instance represents a parser operation (i.e. a transition) and is mapped to a feature vector. We distinguish between *static* and *dynamic* features, where the static features are defined prior to parsing and only depend on the properties of the input buffer, whereas the dynamic features are defined by the

HPSG derivation that is partially built during parsing. Part-of-speech tags and ERG lexical types (supertags) are annotated in a preprocessing stage, and are thus static during parsing.

For the history-based feature model, we assume the formal parser definition in Section 5.2.2, and use the stack position index j for δ and π as defined in Section 5.2.3. We define a set of feature functions to formalize the features used to describe each parsing state (i.e. the *history*, see Section 5.3) for the parser. ι, δ and π are in the following derived from the current configuration of the parser.

- $\beta_{(\iota)}$ is the ιth $W/POS/LT$ triple in the input buffer, where ι denotes the current buffer position.
- BL is the length of buffer β.
- $W_{(\iota)}$ is the ιth word form in β.
- $LT_{(\iota)}$ is the ιth lexical type in β.
- $POS_{(\iota)}$ is the ιth part-of-speech tag in β.
- $LC_{(\iota)}$ is the lexical category tag derived from $LT_{(\iota)}$ (the fields in the lexical type LT were presented in Section 4.1.3).
- $SubCat_{(\iota)}$ is the subcategorization field derived from $LT_{(\iota)}$.
- $IP_{(\iota)}$ and $FP_{(\iota)}$ denote word-initial and word-final punctuation in the word $W_{(\iota)}$, respectively. Specifically, these are extracted using regular expressions to extract all special characters (all characters which are not in the standard English alphabet, or the numbers 0-9) at the beginning and end of the word.
- $\delta_{(j)}$ is the jth edge from the top of the detached edge stack. Conceptually, this edge is a pointer to a disconnected edge in the passive edge stack π.
- $\pi_{(j)}$ is the jth edge from the top of the passive edge stack.
- $l_{(e)}$ is the left daughter of the edge e.
- $r_{(e)}$ is the right daughter of the edge e.
- $h_{(e)}$ is the HPSG head daughter of the edge e, following the Head Feature Principle which was discussed in Section 2.4.2. $h^*_{(e)}$ denotes the head-relation down to the pre-terminal, providing a list consisting of the HPSG category for the head daughter of the edge e, followed by the headed daughters head daughter category etc.
- $H_{(e)}$ is the HPSG rule or lexical type of the edge e.
- $S_{(\delta)}$ denotes the size of the δ stack.

5.3.4 Feature Templates

A *feature template* defines a feature for a given parser configuration, applying the feature functions introduced in Section 5.3.3. These feature templates are used to define the feature model for the parser. A feature can be defined by a single feature template (singleton), or by a conjunction of multiple

feature templates.

The feature templates that are used by CuteForce have been developed under inspiration from other transition-based parsers, especially the HPSG parser described in Ninomiya et al. (2011) and MaltParser (Nivre et al., 2007). 62 feature templates are described in Ninomiya et al. (2011, p. 346), whereas the Dependency parser, MaltParser, has used 14 feature templates (Nivre et al., 2007, p. 107).

The feature templates applied by CuteForce are presented in Table 5.6. We find in total 22 static and 16 dynamic feature templates. For readability we conjoin the feature templates that relate to one specific type of static features to a list. Hence the notation $W_{(\iota-1,\iota,\iota+1)}$ implies that these are three different static feature templates, namely $W_{(\iota-1)}$, $W_{(\iota)}$ and $W_{(\iota+1)}$, where $W_{(\iota-1)}$ is the last processed word from the β stack, $W_{(\iota)}$ is the current word, and $W_{(\iota+1)}$ is a lookahead to the next, currently unprocessed word token on β.

SF	$W_{(\iota-1,\iota,\iota+1)}$,$POS_{(\iota-1,\iota,\iota+1,\iota+2)}$,$LC_{(\iota-1,\iota,\iota+1,\iota+2)}$, ι, $BL-\iota$,$LT_{(\iota-1,\iota,\iota+1,\iota+2)}$,$FP_{(\iota)}$,$IP_{(\iota)}$, $SubCat_{(\iota,\iota+1,\iota+2)}$
DF	$H_{(\pi_{(0)})}$,$H_{(h(\pi_{(0)}))}$,$H_{(h(h(\pi_{(0)})))}$,$H_{(h(h(h(\pi_{(0)}))))}$, $H_{(l(\pi_{(0)}))}$, $H_{(r(\pi(0)))}$,$H_{(h^*(\pi(0)))}$,$H_{(\delta_{(0)}}$,$H_{(h(\delta_{(0)}}$,$H_{(h(h(\delta_{(0)}))}$, $H_{(h(h(h(\delta_{(0)}))))}$,$H_{(l(\delta_{(0)})}$, $H_{(r(\delta_{(0)})}$,$H_{(h^*(\delta_{(0)})}$,$H_{(\delta_{(-1)})}$, $S_{(\delta)}$

Table 5.6: Feature templates for static (SF) and dynamic features (DF).

The feature templates have been designed to represent the relevant linguistic context in order for the oracle to make a precise classification. We see that while the static features (*SF* in Table 5.6) are specific to the input buffer, the dynamic features (*DF* in Table 5.6) refer to the HPSG categories and lexical types in the context of the current state of the parser. There are two larger classes of dynamic feature templates: Templates which refer to the category/lexical types of $\pi_{(0)}$, or an ancestor of $\pi_{(0)}$, and templates which refer to the category/lexical types of $\delta_{(0)}$, or any of its ancestors. Further, $S_{(\delta)}$ refers to the size of the δ stack, and $H_{(\delta_{(-1)})}$ refers to the category/lexical type of the detached edge which appears at the top of the δ stack if a PASSIVE transition is executed, and the δ stack is popped.

In the feature model, most of the atomic features occur in conjunction with other features, and only a few of the features will occur by themselves. A combination of two or more features is necessary to represent the inherent dependence many features have on one another. We will present the final feature model in Section 5.3.6. Obtaining the ideal feature selection is non-trivial, and will be discussed in the next section.

5.3.5 Feature Selection

With the boom in large-scale processing over the last decade, the feature selection task is becoming increasingly complex. While smaller classification tasks may require only a small and rather intuitive selection of features, large-scale classification efforts applying vast amounts of training data may enable a huge set of potentially useful features. Defining the optimal *subset of features* (namely the *feature model*) that are useful for the predictor is hence complex, and has become an emerging field of research. Guyon and Elisseeff (2003) present a comprehensive introduction to variable and feature selection for medium to large datasets. We will discuss how this relates to our own discriminative parsing model.

For a transition-based parser, each instance, i.e. parse configuration, is represented through a feature vector, and this puts a high demand on the feature selection which is used to represent each instance. One advantage with a transition-based parsing algorithm is the ability to utilize a complex feature model for a global context. As we discussed in Section 2.6, a parsing model defined over a packed parse forest is not able to consider global features by the use of feature functions.

Table 5.6 presents the feature templates that we apply for our parser. These feature templates are the atoms of the feature model. They can be applied by themselves (as singletons), or in combination with any other feature(s). Feature combinations are applied if there is an interdependency between two or more features. In our case, this could for instance be a combination of the feature $LT_{(\iota+1)}$ (the next lexical type (LT) on the input buffer) and $H_{(\pi_{(0)})}$ (HPSG rule of the edge at the top of the stack π) – the LT tag alone would provide only limited information to the oracle, since it does not take the local context in the derivation tree into account. By combining these two features, we would however produce a more relevant feature for the instance. This new feature combination, $\langle H_{(\pi_{(0)})}, LT_{(\iota+1)} \rangle$, could again be combined with any other atomic or combined feature to produce a new unique feature. Hence, the number of alternative feature subsets is exponential to the size of the atomic feature template set.

Our goal for the feature selection is to retrieve the feature subset which allows the classifier to perform with the highest possible accuracy. In feature selection techniques we can distinguish between (1) *filter search approaches* which assess the relevancy of the feature subset by inspecting the inherent properties of the data, and (2) *embedded approaches* that interact with a given classifier, reporting a feature relevance according to its contribution to the performance of the classifier in question (the terms are further defined in Guyon and Elisseeff, 2003, Section 4). The latter alternative requires a training and testing iteration for each feature subset which is evaluated, and is typically much more expensive than estimating the relevancy independent of the classifier.

In addition to the search for the optimal feature subset, there are at least two other dimensions: Size of the data set and parameter configuration specific to the given learner. First, if the size of the training data is limited, one would typically use all available data, and design a feature model accordingly. However, in our case we have the entire WikiWoods (47 million sentences, see Section 3.4.3) at our disposal, which provides us with more training data than we are able to utilize. We experienced a computational limitation to the dimensionality (i.e. the number of unique features) that we could train. We were not able to train a model with more than 7.8 million unique features using LibLinear. Considering that there are 274 different classes, it seems we approach the limit with approximately $7,800,000 * 274 \approx 2.13 * 10^{10}$ dimensions. This limit could be reached either by using a very complex feature model, and fewer training instances, or a high number of training instances and a simpler feature model.

Second, the parameter setting constitutes another dimension. The optimal parameter setting of a classifier will vary based on the properties of the feature subset (Daelemans et al., 2003). Such a parameter could be the *cost* parameter found in SVM learners, which regulates the trade-off between allowing training inconsistencies and forcing rigid margins. Alternatively, a parameter defining the minimal number of times a feature value must occur would also in turn introduce a new dimension, because a higher feature frequency threshold would reduce the sparsity of the feature vector, allowing for a higher number of features to be applied in the feature model.

Due to the aforementioned considerations, we have found a fully automated search of an optimal feature subset and overall learner configuration (including parameter setting and size of training data) to be infeasible. To our advantage we have *domain knowledge* – feature selection for a transition-based HPSG parser is not a black box. A search through any feature subset variation is more relevant for classification tasks with anonymous data or data where we have no prior intuitions that we can take advantage of. In our case there exists relevant research for the domain, and we have linguistic intuitions regarding what constitutes a relevant feature and feature combination. Hence, we constructed an ad hoc set of features based on intuitions from other transition-based parsing systems (Ninomiya et al., 2011; Nivre et al., 2007; Sagae and Lavie, 2005, inter alios) and hypotheses of what would be useful considered the search space of the parser and the nature of the HPSG, and started experimenting from this point. The complete selection is presented in Section 5.3.6 below, where we will also discuss the development and evaluation of this feature selection.

5.3.6 Feature Model

Section 5.3.4 introduced the feature templates that are used to define features for the CuteForce Parser. As discussed in the previous section, these 38

feature templates enable an enormous amount of different feature selections, because one atomic feature template can be combined with any number of other feature templates.

Table 5.7 gives an account of the full feature selection used by CuteForce. We have reached this final feature selection through a series of experiments where we have studied the accuracy of the parser when features are added or removed from the feature model, and by using *WEKA data mining software* (Hall et al., 2009) to perform so-called *Single Attribute Evaluation* of the features (this will be discussed in Section 5.3.7). This evaluation has enabled us to design a feature model specifically targeted towards our domain.

Each individual feature in Table 5.7, as it is applied by CuteForce, is captured in angle brackets. In the first row (1) we find the static features. Here we see examples of singleton features (e.g. $\langle FP_{(\iota)} \rangle$) and feature conjunctions (e.g. $\langle W_{(\iota,\iota+1)} \rangle$, which is a combination of the current and the next word item on the buffer). In total there are 38 static features in this category.

In the second row (2) we find the dynamic features. The most important features are the HPSG categories that appear on the top of the δ and π stack, namely $\langle H_{(\pi_{(0)})} \rangle$ and $\langle H_{(\delta_{(0)})} \rangle$. In addition to appearing by themselves, they occur in combination with other dynamic features, typically referring to an ancestor or headed daughter of these rules. Additionally, $\langle S_{(\delta)} \rangle$ appears here, denoting the size of the δ stack. There are in total 22 features in this category.

In the third row (3) we find feature combinations of static and dynamic features. We find two examples of frequently used dynamic and static feature combinations: We see that $LT_{(\iota+1)}$ is often used in conjunction with dynamic features describing the top of the δ and π stack. A lookahead to the next lexical type in the input buffer is also provided among the static features, but it also proved favorable to our system to combine it with features describing the partial HPSG derivation. Further, we see that $\langle BL - \iota, S_{(\delta)} \rangle$ occurs by itself, and in conjunction with other, predominantly dynamic, features. This feature combination describes the size of the δ stack, and the number of unprocessed elements in the input buffer. We find in total 15 complex features combining static and dynamic feature templates.

5.3.7 Feature Evaluation

When designing the final feature model of CuteForce (as presented in Table 5.7), we carried out a large number of experiments on alternative feature models of varying complexities. Further, we used the *WEKA data mining software* (Hall et al., 2009) to perform so-called *Single Attribute Evaluation*. The features (or attributes) are assigned an *Information Gain* and *OneR Attribute Classification Score*. The goal of both evaluation matrices is to assign an individual score to each feature based on its ability to predict the class. The OneR Attribute Evaluation estimates the classification

1	$\langle FP_{(\iota)}\rangle$,$\langle IP_{(\iota)}\rangle$,$\langle BL-\iota\rangle$,$\langle W_{(\iota-1,\iota)}\rangle$,$\langle W_{(\iota-1)}\rangle$,$\langle W_{(\iota+1)}\rangle$, $\langle W_{(\iota)}\rangle$,$\langle W_{(\iota,\iota+1)}\rangle$,$\langle POS_{(\iota-1,\iota,\iota+1)}\rangle$,$\langle POS_{(\iota,\iota+1,\iota+2)}\rangle$,$\langle POS_{(\iota)}\rangle$, $\langle POS_{(\iota,\iota+1)}\rangle$,$\langle POS_{(\iota+1)}\rangle$,$\langle POS_{(\iota-1,\iota)}\rangle$,$\langle LT_{(\iota-1,\iota,\iota+1,\iota+2)}\rangle$, $\langle LT_{(\iota,\iota+1)}\rangle$,$\langle LT_{(\iota,\iota+1,\iota+2)}\rangle$,$\langle LT_{(\iota+1)}\rangle$,$\langle LT_{(\iota)}\rangle$,$\langle LT_{(\iota-1)}\rangle$, $\langle LT_{(\iota-1,\iota)}\rangle$,$\langle LC_{(\iota)}\rangle$,$\langle LC_{(\iota+1)}\rangle$,$\langle LC_{(\iota-1,\iota,\iota+1)}\rangle$,$\langle LC_{(\iota,\iota+1)}\rangle$, $\langle LC_{(\iota,\iota+1,\iota+2)}\rangle$,$\langle LC_{(\iota-1,\iota)}\rangle$,$\langle LC_{(\iota)}, W_{(\iota)}\rangle$,$\langle LC_{(\iota-1,\iota)}, W_{(\iota-1,\iota)}\rangle$, $\langle LC_{(\iota-1,\iota,\iota+1)}, W_{(\iota-1,\iota,\iota+1)}\rangle$,$\langle LC_{(\iota,\iota+1)}, W_{(\iota,\iota+1)}\rangle$,$\langle LC_{(\iota+1)}, W_{(\iota+1)}\rangle$, $\langle LC_{(\iota)}, SubCat_{(\iota)}\rangle$,$\langle LC_{(\iota-1,\iota,\iota+1,\iota+2)}, SubCat_{(\iota-1,\iota,\iota+1,\iota+2)}\rangle$, $\langle LC_{(\iota,\iota+1,\iota+2)}, SubCat_{(\iota,\iota+1,\iota+2)}\rangle$,$\langle LC_{(\iota,\iota+1)}, SubCat_{(\iota,\iota+1)}\rangle$, $\langle LC_{(\iota+1)}, SubCat_{(\iota+1)}\rangle$,$\langle LC_{(\iota-1,\iota,\iota+1)}, SubCat_{(\iota-1,\iota,\iota+1)}\rangle$
2	$\langle H_{(\pi_{(0)})}\rangle$,$\langle H_{(\delta_{(0)})}\rangle$,$\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}\rangle$,$\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(h(\pi_{(0)})))}\rangle$,$\langle H_{(\pi_{(0)})}, H_{(h(h(h(\pi_{(0)}))))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h^*(\pi_{(0)}))}\rangle$,$\langle H_{(r(\pi_{(0)}))}, H_{(l(\pi_{(0)}))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(r(\pi_{(0)}))}, H_{(l(\pi_{(0)}))}\rangle$,$\langle H_{(\delta_{(0)})}, H_{(h(\delta_{(0)}))}\rangle$, $\langle H_{(\delta_{(0)})}, H_{(h(h(\delta_{(0)})))}\rangle$,$\langle H_{(\delta_{(0)})}, H_{(h(h(h(\delta_{(0)}))))}\rangle$, $\langle H_{(\delta_{(0)})}, H_{(h^*(\delta_{(0)}))}\rangle$,$\langle H_{(r(\delta_{(0)}))}, H_{(l(\delta_{(0)}))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h^*(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(h^*(\delta_{(0)}))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(h(\delta_{(0)}))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(h(h(\pi_{(0)}))))}, H_{(\delta_{(0)})}, H_{(h(h(h(\delta_{(0)}))))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(h(\pi_{(0)})))}, H_{(\delta_{(0)})}, H_{(h(h(\delta_{(0)})))}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}\rangle$,$\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(-1)})}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}\rangle$,$\langle S_{(\delta)}\rangle$
3	$\langle LC_{(\iota)}, SubCat_{(\iota)}, H_{(l(\pi_{(0)}))}\rangle$,$\langle LC_{(\iota)}, SubCat_{(\iota)}, H_{(\pi_{(0)})}\rangle$, $\langle BL-\iota, S_{(\delta)}\rangle$,$\langle \iota, BL-\iota, S_{(\delta)}\rangle$,$\langle H_{(\pi_{(0)})}, LT_{(\iota+1)}\rangle$, $\langle H_{(\pi_{(0)})}, BL-\iota, \iota, S_{(\delta)}\rangle$,$\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}, BL-\iota, S_{(\delta)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h^*(\pi_{(0)}))}, LT_{(\iota+1)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h^*(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(h^*(\delta_{(0)}))}, LT_{(\iota+1)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(h(\delta_{(0)}))}, BL-\iota, S_{(\delta)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}, LT_{(\iota+1)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}, LT_{(\iota+1)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}, BL-\iota, S_{(\delta)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(h(\pi_{(0)})))}, H_{(\delta_{(0)})}, H_{(h(h(\delta_{(0)})))}, BL-\iota, S_{(\delta)}\rangle$, $\langle H_{(\pi_{(0)})}, H_{(h(h(h(\pi_{(0)}))))}, H_{(\delta_{(0)})}, H_{(h(h(h(\delta_{(0)}))))}, BL-\iota, S_{(\delta)}\rangle$

Table 5.7: The feature model used by CuteForce. We find the static features in row 1, the dynamic features are in row 2, and combinations of static and dynamic features are in row 3.

accuracy for each individual feature in the feature model given a data set (Holte, 1993). The Information Gain is a score for the expected reduction in entropy that is obtained through each individual feature (Breiman et al., 1984). By analyzing the differences in performance when applying alternative feature models, we have reached a final feature selection that maintains a good balance between granularity and scalability for our domain. Table 5.8 provides a list of the top 10 ranked features by their Information Gain

score, with corresponding OneR Attribute Evaluation scores.

Feature	**InfoGain**	**OneR**
$\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(h(\delta_{(0)}))}\rangle$	3.51547	74.8325
$\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}\rangle$	3.51079	71.3444
$\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}\rangle$	3.50916	73.0436
$\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(\delta_{(-1)})}\rangle$	3.49018	74.9611
$\langle H_{(\pi_{(0)})}, H_{(h(h(\pi_{(0)})))}, H_{(\delta_{(0)})}, H_{(h(h(\delta_{(0)})))}\rangle$	3.42753	74.6077
$\langle H_{(\pi_{(0)})}, H_{(h(h(h(\pi_{(0)}))))}, H_{(\delta_{(0)})}, H_{(h(h(h(\delta_{(0)}))))}\rangle$	3.42188	74.4264
$\langle H_{(\pi_{(0)})}, H_{(h^*(\pi_{(0)}))}, H_{(\delta_{(0)})}, H_{(h^*(\delta_{(0)}))}\rangle$	3.2022	73.1839
$\langle H_{(\pi_{(0)})}, H_{(\delta_{(0)})}, BL-\iota, S_{(\delta)}\rangle$	3.20071	72.3323
$\langle H_{(\pi_{(0)})}, H_{(h(\pi_{(0)}))}, H_{(\delta_{(-1)})}\rangle$	3.18703	65.8997
$\langle H_{(\pi_{(0)})}, H_{(h^*(\pi_{(0)}))}, LT_{(\iota+1)}\rangle$	3.09279	68.152

Table 5.8: The ten highest ranked features based on the Information Gain Scores, with corresponding OneR Attribute Evaluator values, computed by *WEKA data mining software*.

The outcome of the evaluation in Table 5.8 agrees well with our expectations. The feature template $H_{(\pi_{(0)})}$, describing the HPSG category on the top of the π stack is present in all of the highest ranked features, and the corresponding template for the category for the top of the δ stack ($H_{(\delta_{(0)})}$) is present in the eight highest ranked features. The other feature templates describe the headed ancestors of $\delta_{(0)}/\pi_{(0)}$, except from $LT_{(\iota+1)}$ (describing the next lexical type on the input buffer), $S_{(\delta)}$ (the size of the δ stack), and $BL-\iota$ (the number of unprocessed elements on the input buffer).

The scoring provided by the individual feature evaluation was useful for identifying features which did not contribute to the overall performance of the classifier. However, a feature with a low contribution estimate score could in some cases prove vital to the parser. These could be features that are useful only in limited configurations. A typical example of this is a feature that specifies the punctuation of the current word in the input buffer ($\langle FP_{(\iota)}\rangle$ and $\langle IP_{(\iota)}\rangle$). Although this feature contributes very little in most parse configurations, it is crucial when determining the lexical punctuation rules.

Our final feature selection is tuned in harmony with the machine learning solver we chose, the parameter setting and the size of the training data. A training cycle takes about 20 hours, which puts severe constraints on our ability to do large-scale automatic feature selection (Guyon and Elisseeff, 2003 discusses automatic feature selection, e.g. through the use of the *backward elimination approach*). Although we feel confident our final feature selection and overall configuration is near-optimal for this classification task, the hypothetical dimensions that could be explored seems unquantifiable, and this coupled with a costly training cycle suggests that further

optimizations could be obtained through additional experimentation and tweaking.

5.4 Conclusion

In this chapter we have presented the CuteForce Parser, which enables transition-based deterministic HPSG parsing. We have focused on two fundamental aspects:

1. The transition-based algorithm that enables HPSG parsing for the DELPH-IN framework, specifically for the English Resource Grammar (ERG)
2. The *Oracle* that navigates the search space by predicting the parser actions

We have shown a transition-system that can generate any given ERG derivation through a unique sequence of transitions. The transition system and the deterministic configuration ensure a linear time-complexity, enabling highly efficient parsing for a unification-based grammar. Further, we have discussed the oracle that is used to navigate the search space. While the oracle of a deterministic parser is restricted from taking the full context of the sentence into account, this is balanced by the fact that our framework facilitates the use of global and local feature functions for each decision point, allowing for the use of syntacto-semantic information that would not be available in a chart parsing-based formalism.

The performance of the CuteForce Parser will be evaluated in Chapter 7, where we will provide statistics for the parser and compare it to existing parsers for the same grammatical framework.

Chapter 6

Near-Deterministic Parsing

The main objective for this thesis has been to explore the field of deterministic HPSG parsing. In this chapter we will however turn our attention to *near-deterministic parsing* (near-deterministic parsing for a transition-based framework was introduced in Section 2.6.1). In the following, we will present an innovative backtracking strategy which can be employed as an overlay to a transition-based deterministic parser, in our case specifically for CuteForce. This method, dubbed *Optimistic Backtracking* (also presented in Ytrestøl, 2011b), can be used to gauge the usefulness of backtracking as a fallback strategy when the deterministic parsing of a sentence ends in parsing failure. In itself the algorithm is heavily constrained to minimize computational complexity – this can however be relaxed if initial testing indicates that a more permissive backtracking regime would benefit the parser.

This chapter will present the Optimistic Backtracking algorithm and how it is adapted to CuteForce. Further, the scope of the backtracking problem will be elaborated. The results and the evaluation for CuteForce when applying the backtracking overlay will be presented and discussed in the following chapter, Section 7.6.

6.1 Motivation for Near-Deterministic Parsing

A deterministic parsing algorithm like the one we proposed in the preceding chapter is from a computational standpoint maximally efficient – it does not waste time pursuing any search paths that will not contribute to the final parse derivation. However, from the last chapter we remember that due to the hard constraints in HPSG and other unification-based grammars, a deterministic parser may arrive at a state from which there are no valid continuations. These *parsing failures* (further discussed in Section 6.4.4) arise when the parsing oracle makes an incorrect decision at an earlier stage which immediately or further down the search path will lead the parser to an *invalid state*, e.g. a configuration from which there are no valid transi-

tions. The *parsing mode* (discussed in Section 5.2.5) of CuteForce determines whether the parser may arrive at such a state, resulting in a parsing failure. When parsing in unrestricted parsing mode we would never arrive in an invalid state — the need for recovery from an invalid state could only arise when parsing with a CFG filter or when applying unification.

This arrival at a parsing configuration with no valid continuations could be said to resemble the cognitive state we find ourselves in when misinterpreting a sentence as we read/hear it for the first time. As we outlined in Section 2.6, garden paths are the canonical example of this phenomenon. It would be plausible to assume that a sentence like (1) would cause problems for a deterministic parser, because the analysis of the sentence is built word-by-word, using assumptions about how the sentence will end.

(1) The horse raced past the barn fell.

As we briefly discussed in Section 2.6, when processing the word *raced* in (1), it would seem more plausible for a deterministic parser that *raced* constitutes the head verb of an active sentence, rather than being a part of a reduced relative clause, unless the training data is unusually biased towards such reduced relative clause constructions. Although a lookahead window where the oracle takes the immediately following tokens or POS tags/supertags in the input buffer into consideration could accommodate some proportion of these cases, this lookahead window is typically limited to no more than the four following words or tags (the parsers presented in Ninomiya et al., 2011 and Nivre et al., 2007 apply a lookahead window of the three next buffer items), and one can easily imagine corner-cases where even larger lookahead windows still would not suffice to capture the *temporary ambiguity*[1] that may be present when processing the sentence.

The ability to reevaluate a previous decision made by the oracle is prohibited in a deterministic parsing scheme. Optimistic Backtracking provides a near-deterministic method designed to locate the incorrect parser decision in the sequence of transitions that was executed up until parsing failure. The backtracker is employed if the parser reaches a configuration from which there are no valid transitions, which would be a state from which a deterministic parser would fail to produce a parse.

The primary goal of the Optimistic Backtracking method is to identify and replace this presumably incorrect parser decision with the correct parser transition, and resume parsing from this state. However, even in cases where the backtracker fails to identify and substitute the first incorrect parsing decision with the correct one, we still expect that the backtracker will

[1] A phrase more frequently found in psycholinguistics, describing utterances where readers typically "assign a syntactic analysis (the syntactically simplest one) rather than either delaying analysis or developing and maintaining multiple structural analyses." (Rayner and Frazier, 1987, p. 657)

contribute to increasing the coverage by finding a valid parse, even though it may not be identical to the gold standard.

6.2 Related Near-Deterministic Approaches

In Section 2.7 we gave an overview of prominent deterministic and near-deterministic parsers over the past decade. As we remember, a few of these, most prominently the HPSG parser developed by Ninomiya et al. (2011), provide both deterministic and near-deterministic parsing algorithms. Further, beam-search strategies are pursued for a range of grammatical frameworks, e.g. CCG (Zhang and Clark, 2011a), dependency grammar (Zhang and Clark, 2011b) and CFG (Collins and Roark, 2004).

The near-deterministic approaches for CFG, CCG and dependency grammar are typically motivated from an accuracy perspective, where the assumption that maintaining multiple partial analyses through the use of beam-search will contribute to more precise parsing results than committing to one single search path in a deterministic parsing scheme. The parsing for unification-based grammars like HPSG provides another incentive for investigating near-deterministic methods, namely the risk of parsing failures, as mentioned in Section 6.1. This will be further discussed in the following section.

6.2.1 Near-Deterministic Parsing with HPSG

Due to a common grammatical framework, the HPSG parser presented in Ninomiya et al. (2011) is of particular interest to us. Ninomiya et al. (2011) propose two near-deterministic strategies: Beam Search and Backtracking. These approaches, presented in Section 2.7.4, provided a performance boost for the parser at the expense of higher running times – the best-scoring result was obtained by using a very wide beam, something which resulted in an average runtime of approximately 14 seconds per sentence. Further, they reported a parsing failure reduction of 51.9 % for their backtracking approach, and 55.1% and 90.5 % for their beam search method (normal and wide beam width) compared to the deterministic alternative when applying normal unification.

The findings in Ninomiya et al. (2011) suggest that near-deterministic approaches can be a fruitful strategy for mitigating parsing failures that can arise when parsing with unification-based frameworks. Inspired by their findings, we have designed an innovative backtracking algorithm which can be used to assess the potential advantages of applying backtracking as a means to improve a deterministic parser. This approach will be presented in the following sections.

6.3 Backtracking for the CuteForce Parser

Optimistic Backtracking is a method that can be implemented as an overlay to a deterministic transition-based parser in order to evaluate the parser in near-deterministic mode. Before we turn to the specific details of this backtracking approach, we will present the requirements and considerations that a backtracker for CuteForce will have. We chose to exemplify with CuteForce since we have introduced this parser in Chapter 5, but we assume that these considerations would largely be valid for most other deterministic transition-based parsers that can experience parsing failure (*parsing failure* in the context of near-deterministic parsing is discussed in Section 6.4.4).

A transition-based parser maintains a data structure, the parser configuration (c_n), which can be derived from the n transitions in the transition sequence T which precedes c_n (this is elaborated in Section 5.2.2). The transition sequence is defined by the oracle, and we formalized this sequence in Section 5.2.3:

$$T = \underset{t \in N(c_0)}{\arg\max}\, P(t \mid c_0), \underset{t \in N(c_1)}{\arg\max}\, P(t \mid c_1), \ldots, \underset{t \in N(c_n)}{\arg\max}\, P(t \mid c_n)$$

For the remainder of the chapter we will refer to a given transition n in T as t_n, rather then the more verbose $\arg\max_{t \in N(c_n)} P(t \mid c_n)$ for readability.

The backtracker is launched when the deterministic parsing of a sentence fails, after the transition sequence T has been executed (a parsing failure will be discussed further in Section 6.4.4). The parsing failure follows indirectly because the parsing oracle failed to identify the correct transition as t_e at a configuration c_e. We will use the index $_e$ throughout this chapter to refer to the configuration where the parsing oracle committed an ***e***rror. Choosing an incorrect search path implicates that the correct analysis will not be retrieved, and although this does not automatically lead to a parsing failure, the only way we can be certain a parsing failure will not occur, is by choosing the correct path. Hence the backtracker will try to identify the incorrect search path that was chosen, select the correct path, and continue parsing from this point.

We know that the correct transition is among the transitions ranked below the highest ranked transition in $N(c_e)$.[2] All configurations executed after this point are inconsistent with the gold standard, and the correct HPSG analysis cannot be generated after the initial incorrect assumption of t_e, regardless of the search path that is selected after this point.

The responsibility of the backtracker is hence twofold:

[2]Except for events where the CFG approximation filter incorrectly removes the correct transition for a given parsing state, because the rule projection that a correct transition would yield, is not captured among the valid CFG projections.

1. Identify the first transition t_e in T that deviates from the gold standard.
2. Choose the correct next transition $t \in N(c_e)$, which can be any of the transitions excluding $\arg\max_{t \in N(c_e)} P(t \mid c_e)$

6.3.1 Scope of the Backtracking Problem

Upon parsing failure, the search space for the backtracker is defined by the twofold task presented above. The theoretical search space equals the sum of available, *unpursued* transitions for each configuration c_c that has lead up to the configuration c_f, in which parsing failed (the index $_f$ will be used throughout the chapter to refer to the configuration where parsing failed, and backtracking was initiated). For the first backtrack attempt, all candidates in $N(c_c)$ are eligible, except the highest ranked, as we outlined in task (2) in Section 6.3 above. If we use the function $length()$ to denote the number of transition candidates for a configuration c_c, the search space, i.e. the pool of eligible alternative search paths for a backtracker that fails in configuration c_f is:

$$\sum_{i=0}^{f-1} length(N(c_i)) - 1$$

In this pool there is exactly one transition candidate that must be retrieved in order for the backtracker to succeed. When correctly identifying the configuration c_e from which the parsing oracle incorrectly assumed that t_e was the correct parsing transition, we need to identify the correct transition in the set $N(c_e)$, whereupon this transition is executed and parsing can resume from the new configuration c_{e+1}.

Narrowing the Search Space Optimistic Backtracking will not consider all the theoretical candidates for each backtracking attempt. Our method will follow the intuitions in Ninomiya et al. (2011), and choose the second-highest ranked transition in $N(c_e)$, after having identified the presumably incorrect transition in T. Considering that the parsing oracle has already ranked each valid transition for a given configuration, we make the assumption that the runner-up transition ranked by the parsing oracle is the most likely correct transition, given that the parser's initial assumption was incorrect.

Even though we leave this decision to the parsing oracle, there are a number of different factors that influence the accuracy of the backtracker. First, the permissiveness of the filter (CFG approximation or unification) determines when the parser reaches an illegal state – a more permissive filter could possibly allow for more mistakes before it reaches a dead end, thus increasing the amount of candidates. A second plausible factor is the quality

of the parsing oracle. Ironically, a good parse model is likely to make more 'intelligent' mistakes than a worse-performing parse model, something that in all likelihood would make the mistakes less obvious to recognize.

6.4 Optimistic Backtracking

If the backtracker is launched unrestrictedly every time a parsing failure is encountered, parsing will have exponential time-complexity. Hence, a restriction on the number of backtracking attempts needs to be defined. Ninomiya et al. (2011) defines a maximum number of attempts for each sentence. If this number is reached, and the parser still encounters a parsing failure, the parsing fails for this given sentence.

Rather then statically defining a maximum number of backtracking attempts, we propose an alternative restriction to the number of backtracking invocations: The first time the backtracker is invoked (in configuration c_f), all transitions in T are eligible backtracking candidates. If t_e in T is identified and reevaluated, parsing continues from c_{e+1}. If parsing again fails, the backtracker will optimistically assume that its first backtrack was correct, and that c_e indeed is following the correct search path. Hence, the eligible backtracking candidate are the transitions executed after this point (t_{e+1},t_{e+2},...,t_{f-1}). From this follows that if the parser fails after f transitions, it can have maximally $f-1$ backtracking attempts. Used as an overlay for CuteForce, these restrictions ensures a worst-case quadratic time-complexity for the (near-deterministic) parser. However, we observe that on average the parsing time merely doubles when parsing with the Optimistic Backtracking, compared to a deterministic parsing scheme. The parsing times, together with the parsing results are discussed in Section 7.6. By enforcing a constrained backtracking regime, we aim to preserve a highly efficient parsing strategy. Having allowed the parser to backtrack unrestrictedly, we could theoretically have reached close to 100 % coverage, but that would imply an exhaustive or near-exhaustive search, and such a parsing strategy could be more efficiently carried out through the use of a conventional chart parser (i.e. use of dynamic programming).

6.4.1 Motivation

There are as we see it two strong motivations for employing a dynamic constraint to the number of possible backtracks: First, it ensures that the maximum number of backtracking attempts varies with the length of the transition sequence, which again is directly related to the input length. This reflects the intuition that parsing failures are more likely to occur in longer sentences, and because of the increased search space, it is expected that more backtracking attempts are required for longer sentences than sentences of shorter input length.

Second, this method will automatically ensure that the backtracker does not enter a loop by investigating search paths it has previously entered. Without a built-in restriction like the 'optimistic' constraint presented above, additional heuristics would have been called for that would keep track of earlier search path, something that would have increased the complexity of the backtracking task.

6.4.2 Data Structure

As an overlay for a deterministic transition-based parser, Optimistic Backtracking requires some augmentation to the data structure. In Section 5.2.3 we defined a parser configuration $c = (\iota, \pi, \delta)$ for the deterministic CuteForce Parser. For near-deterministic parsing using Optimistic Backtracking, this data structure could be expanded to the 5-tuple:

$$c = (\iota, \pi, \delta, T, t2) \tag{6.1}$$

T holds the transition sequence which has been executed up until a given configuration c_c. This sequence can be used to derive any parser configuration that lies prior to the current parser configuration c_c by replaying the transitions up to a given point. $t2$ denotes the parsing oracle's second-highest ranked transition for each parse configuration c.

When the backtracker re-evaluates the e-th parser decision, i.e. the parser decision which it assumes to be incorrect, the parser will revert to c_e by replaying the $e-1$ first transitions in T, whereupon the configuration c_e is reached. Then the transition $t2$ in c_e will be executed, and parsing will resume from this point (c_{e+1}).

Finally, we require an index L referring to the last transition in T (t_L) that was re-evaluated. According to our 'optimistic' assumptions, we consider the configuration c_{L+1} to be following the correct search path after t_L has been reevaluated, and only the transitions occurring after this point (t_{L+1},t_{L+2},...,t_{f-1}) are eligible for backtracking. At the beginning of a parse, L is initialized to -1. L is not a part of c, because it is not relative to each configuration, but should rather be considered a global variable.

6.4.3 Ranking

Ninomiya et al. (2011) presents a backtracking approach along with a beam-search method for transition-based HPSG parsing. Upon parsing failure, they outline several approaches that can be used to identify the incorrect transition in T, hence the point to which we want to revert:

> There are several metrics to choose a previous state as a restarting point, e.g., the smallest probability for transition, the smallest difference or ratio of probabilities for the top-ranked and

> second-ranked transitions, or the largest entropy in the transition candidates. We used the difference in probability between the top-ranked and the second-ranked transition candidates as the metric in this study, and the previous state with the smallest difference in probability was selected. (Ninomiya et al., 2011, p. 348)

Common for all of the methods outlined above is that they only consider the transition sequence T and the probability distribution in the set of valid transitions for each transition ($N(c_c)$). They do not however consider the partial analysis being derived, i.e. the *global context* for the transitions in the sequence T.

To enable a backtracking approach that can consider a global context, we propose a method that treats the selection of backtracking points as a ranking problem. Doing so, we are able to rank each individual parser operation in T, not only based on the parser oracle's local probability distribution in the set $N(c_c)$, but on the entire sequence of configurations c_c that is generated throughout the parse, e.g. including features that consider global information that relates to the partial HPSG derivation that was built up to the point of parsing failure, and the input buffer for the sentence.

It is important to remember the premise of backtracking, namely that the aim is to identify the *first* transition t_e in T that deviates from the gold standard. All transitions beyond the configuration c_e are irrelevant, but a backtracker that only considers the local context c_c for each candidate backtracking point has no means to capture that a transition t_c that occurred after the initial parsing mistake t_e is an incorrect backtracking point, regardless of the probabilistic distribution for the transitions in $N(c_{e+i})$ when $i > 0$. This intuition may however be captured when the task of identifying the backtracking point is treated as a ranking problem. To our knowledge, *Optimistic Backtracking* is the first backtracking algorithm that investigates ranking for this task.

Quite intuitively, utilizing global information seems attractive for multiple reasons: Again consider sentence (1) (*The horse raced past the barn fell.*). It is when the second verb (*fell*) is encountered that we are likely to reevaluate our original assumption, namely that *raced* is the head verb of the sentence. Cognitively speaking, that *fell* indeed is a verb is surely relevant information for reconsidering whether *raced* is the head verb of the sentence. The ranking-based backtracking framework we propose here enables us to make use of such global syntacto-semantic information in the backtracking, something which distinguishes our approach from conventional backtracking approaches that are restricted to only consider the transition sequence T.

Further, treating backtracking as a ranking problem could be vital for classification-based parsing oracles that do not yield a probability score in the ranked list of candidate parser transitions ($N(c_c)$). Although *Optimistic*

Backtracking does use a transition-specific pseudo-probability as a feature in the feature model (we will return to this in Section 6.5), this is only one of many features employed by the ranker.

6.4.4 Parsing Failure

The transition-based HPSG parser presented in this thesis, CuteForce, can operate in unrestricted mode, HPSG unification mode and in CFG approximation mode. Both the unification filter and the CFG approximation guide the parser by eliminating transitions that violate the constraints defined by these parsing modes (Section 5.2.5 discusses the various parsing modes for CuteForce). If the parser arrives at a configuration from which all further transitions are prohibited, parsing fails for the current sentence. This is the point-of-failure that is utilized by the Optimistic Backtracking algorithm as a starting point for backtracking.

A parsing failure may also occur when applying the Optimistic Backtracking algorithm. This will happen if a parsing failure is encountered in the configuration c_f, and $f = L + 1$, implying that the last backtracking point was to c_{f-1}. In this case, there are no eligible backtracking candidates in T, and the parsing of the sentence is terminated.

The optimistic backtracking algorithm could be applied to transition-based deterministic parsers of other formalisms, given that a cue that triggers backtracking can be defined, equivalent to a parsing failure in CuteForce.

6.4.5 Less Constrained Algorithm

The *Optimistic Backtracking* method is designed to optimize performance and create a minimal amount of overhead in terms of parser complexity. The strong emphasis on efficiency results in a heavily constrained algorithm. Following the 'optimistic' intuition, if a parsing failure occurs, the backtracker must operate with a 100% accuracy each time it is invoked if the parser is to have a chance of retrieving the correct parse.

As we remember from Section 6.4.4, parsing with optimistic backtracking fails if a parsing failure is encountered in the configuration c_f, and the previous backtracking point was c_{f-1}. Having relaxed the 'optimistic' premise, namely to not trust all previous backtracks to be correct, we could have started over again with a new backtracking, e.g. by keeping a priority queue of candidate incorrect parser decisions – pursuing a different backtracking decision if the highest ranked decision did not result in a successful parse.

Such a relaxation of the Optimistic Backtracking constraints would be likely to increase both the parser's coverage and accuracy. However, an overly liberal backtracking regime would be unfavorable from a complexity perspective. We also feel that applying a very liberal backtracking algo-

rithm could overshadow the insights and advantages of a parsing algorithm that was designed as a deterministic parsing approach. Since deterministic parsing has been the main focus of this thesis, the implementation of the *Optimistic* backtracking overlay serves primarily as an indicator of if and how near-deterministic approaches would be of benefit to our parser.

6.5 Ranking Model

Section 6.4 presented the constraints of the backtracking algorithm that ensures that CuteForce runs with a worst-case quadratic complexity when applying the Optimistic Backtracking overlay. In addition to this algorithm, we require a method that aims to identify the first transition in T that deviates from the gold standard, as we discussed in Section 6.3. In the following, we will present a ranking-based approach that ranks each candidate transitions in T according to their likelihood of being the first incorrect parser decision executed by the parsing oracle. After having identified a transition t_e in T, we retrieve configuration c_e, and execute transition $t2$ in c_e, arriving at a new configuration c_{e+1}.

In Section 6.6 we introduce an alternative backtracking approach. It does follow the 'optimistic constraints' as defined in Section 6.4. However, rather than using ranking, it applies the local context of each ranked list of transitions $N(c_e)$ in T when determining each backtracking point. This method for retrieving the backtracking point corresponds to the method proposed in Ninomiya et al. (2011), and the accuracy of these two strategies will be presented and evaluated in Section 6.7.

6.5.1 Candidate Transition Ranking

If a parsing failure occurs in configuration c_f, we know that an incorrect decision by the parsing oracle was made prior to this configuration. Since T in c_f stores all transitions leading up to c_f, we can further conclude that the incorrect parsing decision is found here. However, honoring the optimistic constraint, we will only consider the transitions $t_{L+1}, t_{L+2}, \ldots, t_{f-1}$, i.e. the transitions that were executed after the last backtracking attempt, defined by the global variable L. If this is the first backtracking attempt, all parsing decisions in T will be considered, since L's initial value is set to -1.

Each of these candidate transitions are mapped to individual feature vectors that provide information about the parsing decisions. The task for the ranker is to identify the first incorrect transition in the sequence of transitions T. The feature vector maps 29 features in total for each candidate transition. The ranker is trained as a linear SVM using SVM^{rank}, and we refer to Joachims (2006) for a further discussion of the underlying training and ranking algorithm.

As mentioned in Section 6.3, using a ranker allows us to combine local transition-specific information, together with global syntacto-semantic information from the (partial) HPSG derivation to describe each candidate transition in a feature vector. The exact features which are used to describe each transition will be presented below. As in the previous chapter, we will introduce the feature functions, the feature templates and finally the feature selection which is used by the ranking model for the backtracker.

Feature Functions

The list of transitions $t_{L+1}, t_{L+2}, \ldots, t_{f-1}$ from T comprises the candidate transitions that are ranked by the backtracking ranking model upon parsing failure in configuration c_f. We are not only ranking the actual transitions in T, but each configuration c_{L+1},c_{L+2},... leading up to the parsing failure in c_f. We will use a Java-like operator (.) to access the individual data structures which are stored in c. Hence, the parser transition that was executed in c_c is $c_c.t_c$. The second-highest ranked transition is $c_c.t2$, the π stack is $c_c.\pi$, ι for the configuration c is $c_c.\iota$ etc.

First we need to define two indexes. These indexes will be used as address functions for the parse configurations c, and also as features in the feature model:

- f is the index for the parse configuration in which the parsing failure occurred (c_f). The configurations prior to c_f is hence c_{f-1},c_{f-2}, etc, down to the initial parse configuration c_0.
- c is the index for the configuration number which is being mapped to a feature vector. When a parsing failure occurs in configuration c_f, all eligible transitions leading up to c_f are subject for reevaluation. These eligible transitions are accessed through the parse configuration in which they were executed. Hence, c_c is in the following used to describe the parser configuration in which the transition that was executed is subject for re-evaluation.

The feature functions are defined below. We see that all the functions are described relative to a configuration c. The configuration c is in its turn either relative to the current configuration which is mapped to the feature vector ($_c$), or relative to the last configuration before parsing failure ($_f$).

- $P(c_c.t_c)$ is the pseudo-probability for the transition t_c that was executed in configuration c_c. This score is provided by the parsing oracle, and is the dot product of the feature vector and the weight vector for this transition, as computed by the parsing oracle.
- $P(c_c.t2)$ is the pseudo-probability for the second-highest scoring transition ranked by the parsing oracle for the configuration c_c.

- $C(c_c.t_c)$ is the name of the *transition category* for the transition $c_c.t_c$. The transition name includes the main transition (e.g. *PASSIVE*), and the parse label for the parameterized parse transition (e.g. *sp-hd_n_c*). There are in total 274 unique transition categories, and this is further discussed in Section 5.2.3.
- $PP(c_c)$ is an abbreviation for the probability proportion of the second-highest scoring probability $P(c_c.t2)$ over the sum of the probabilities $P(c_c.t2)$ and $P(c_c.t_c)$: $\frac{P(c_c.t2)}{P(c_c.t2)+P(c_c.t_c)}$. In practice this score is used to determine the confidence level of the parsing oracle. The higher this score is, the less confident the parsing oracle was that $c_c.t_c$ should be executed, as opposed to $c_c.t2$
- $LT(c_f.\iota)$ is the last lexical type in the input buffer that was processed before parsing failure (the $LT()$ feature function is identical to the previous definition in Section 5.3.3 for the feature functions for the deterministic CuteForce setup)
- $POS(c_f.\iota)$ is the last POS tag in the input buffer that was processed before parsing failure (the $POS()$ feature function is identical to the previous definition in Section 5.3.3)
- $H(c_c.\pi_0)$ is the HPSG category/lexical type on top of the π stack in configuration c_c
- $H(c_c.\delta_0)$ is the HPSG category/lexical type on top of the δ stack in configuration c_c

Feature Templates

The 15 feature templates applied by the ranking model are presented in Table 6.1. The feature selection used by the ranker employs features that can roughly be divided in four groups. First, the transition-specific features provide information on the nature of the candidate transition and surrounding transitions. Here we also have features related to the pseudo-probability of the transition (provided by the parsing oracle), and the parser's confidence in the transition it ranked highest.

Second we have features related to the last token that was processed by the parser before it reached an invalid state, and information on the incomplete HPSG derivation that was built at that state. As atomic features, they will remain constant for all candidate transitions that are being ranked. Hence, we will only find these templates in conjunction with other feature templates which are relative to the current parser configuration c_c.

Third, we have features concerning the preliminary HPSG derivation and the lexical types from the buffer relative to the candidate transition which is being described.

The final group provides positional information. $_c$ refers to the candidate transition index that is being mapped to the feature vector. $_f$ refers to the total number of transitions that was executed upon parsing failure.

Table 6.1 lists the feature templates according to these four classes.

Transition-specific	$C(c_c.t_{c-1})$,$C(c_c.t_c)$,$C(c_{c+1}.t_{c+1})$ $PP(c_c)$,$P(c_c.t_c)$
Last token	$LT(c_f.\iota)$,$POS(c_f.\iota)$,$H(c_f.\pi_0)$
HPSG derivation	$H(c_c.\pi_0)$,$H(c_c.\delta_0)$,$LT(c_c.\iota-1)$ $LT(c_c.\iota)$,$LT(c_c.\iota+1)$
Position	c,$f-c$

Table 6.1: Feature Templates for the ranking model.

Feature Model

The final feature selection, i.e. the feature model for the backtracking ranker, was designed through an evaluation of various feature subsets. By evaluating the accuracy of the ranker when utilizing various feature subsets, we arrived at a feature selection consisting of 29 features, constructed through the use of the 15 feature templates presented in Table 6.1. Table 6.2 presents the feature selection which is applied by the ranker.

1	$\langle C(c_c.t_{c-1})\rangle$,$\langle C(c_c.t_c)\rangle$,$\langle C(c_{c+1}.t_{c+1})\rangle$
	$\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1})\rangle$
	$\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1}), P(c_c.t_c)\rangle$
	$\langle C(c_c.t_c), PP(c_c), P(c_c.t2)\rangle$,$\langle C(c_c.t_c), P(c_c.t2)\rangle$
	$\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1}), P(c_c.t_c), PP(c_c)\rangle$
	$\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1}), PP(c_c)\rangle$
	$\langle C(c_c.t_{c-1}), P(c_c.t_c), PP(c_c)\rangle$,$\langle C(c_c.t_c), PP(c_c)\rangle$
	$\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1}), P(c_c.t_c), PP(c_c)\rangle$
	$\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1}), c, f-c\rangle$,$\langle C(c_c.t_c), c\rangle$
	$\langle C(c_c.t_c), c, f-c\rangle$,$\langle PP(c_c), f-c\rangle$,$\langle PP(c_c), c, f-c\rangle$
	$\langle P(c_c.t_c)\rangle$,$\langle PP(c_c)\rangle$,$\langle P(c_c.t_c), PP(c_c)\rangle$
2	$\langle LT(c_f.\iota), C(c_c.t_c)\rangle$,$\langle POS(c_f.\iota), C(c_c.t_c)\rangle$,$\langle H(c_f.\pi_0), C(c_c.t_c)\rangle$
3	$\langle H(c_c.\pi_0), H(H(c_c.\delta_0))\rangle$,$\langle H(c_c.\pi_0)\rangle$
	$\langle LT(c_c.\iota-1), LT(c_c.\iota)$,$LT(c_c.\iota+1)\rangle$
4	$\langle c\rangle$,$\langle f-c\rangle$,$\langle c, f-c\rangle$

Table 6.2: The feature model used by the backtracker.

The first group (1) in Table 6.2 holds the transition-specific features. We see that the feature $\langle C(c_c.t_{c-1}), C(c_c.t_c), C(c_{c+1}.t_{c+1})\rangle$ is a list of the current transition category, in conjunction with the category immediately preceding or following the current transition category. The can be seen as an n-gram (specifically a 3-gram) over the transitions that were executed in the context of the current parse configuration c_c that is mapped to a feature vector. In

this group we also find features relating to the pseudo-probability computed by the parsing oracle for a transition (accessed with the $P()$ function). This pseudo-probability can be a measure of the uncertainty of the CuteForce parsing oracle when committing to this specific transition. We see that the probability-related features P and PP occur as singletons, describing probability and confidence estimates for the specific transition that was executed for c_c, and they occur in conjunction with the features that extract transition categories for the current parsing configuration, and the surrounding context, c_{c-1} and c_{c+1}. The intuition of these feature conjunctions is to capture sequences which frequently proved indicative of an incorrect parser decision. The probabilities in the feature model are represented as interval sequences ($0.0 - 0.1$, $0.11 - 0.2$, etc.) with nominal categories, i.e. a predefined list of feature values.

In the alternative backtracking strategy presented in Section 6.6, the confidence metric PP is the only element taken into consideration when selecting a backtracking transition candidate. Further, this class contains features referring to the transition number c, and the relative positioning of the current transition compared to the length of the list of applicable transition candidates ($f - c$). The transition's position in the sequence can be useful because the distribution of transition categories are not uniform throughout a parsing sequence, e.g. a PASSIVE(sb-hd_mc_c) transition, instantiating a Subject+Head main clause would be expected to appear at the very end of a transition sequence T. If it appears in any other range of the sequence, this may indicate that the parsing decision is incorrect. By combining the positional information through the features (c and $f - c$), we are able to express this information, e.g. with the feature $\langle C(c_c.t_c), c, f - c\rangle$.

The second group (2) combines the feature template for the current transition candidate with syntacto-semantic information concerning the parser configuration at the time of parsing failure (c_f). The intuition for this was discussed in Section 6.4.3, where in the event of a garden path in Sentence (1), we encounter the head verb at the very end of the sentence, at which point a deterministic parser would be likely to fail if it had mistakingly identified *raced* to be the head verb of the sentence. Considering that an incorrect parser transition is the direct cause for the parsing failure in c_f, conjoining information about the current transition and the parsing configuration at parsing failure is one of the fundamental intuitions behind the use of ranking in backtracking.

The third group (3) refers to the HPSG categories/lexical types on top of the δ/π stack. Originally, our intension was to combine these features with the feature describing the current transition $C(c_c.t_c)$, hence aiming to capture reoccurring parsing mistakes conducted by the parsing oracle, e.g. the under- or overgeneration of a PASSIVE transition. However, our testing during the development of the backtracker suggested that this feature conjunction had a negative effect on the classifier, possibly because it gen-

erated data sparseness. Hence, the third group of features serve currently to describe the top of the δ/π stack, and the lexical types in the context of the current transition.

The fourth group (4) contains the positional feature templates which refer to the transition number of the current transition in the sequence T, and to the total number of transitions in T before the parsing failure occurred. Although these features mostly occurred in conjunction with transition-specific features, it proved useful to include them in the feature model as individual features.

6.5.2 Training Data

The ranker requires a certain amount of training data in order to perform optimally. Providing training material for the backtracking oracle is relatively straightforward. Training data is collected by parsing in deterministic mode; every time the parser reached a parsing failure, it would record the transition candidates which are subject to backtracking, and label the parsing decision that deviated from the gold standard ($c_e.T_e$) as the correct case. Further it will repair the incorrect parsing decision in c_e, and continue parsing from this point. Hence, in this configuration, the parser is parsing with an optimal backtracker that always identifies and repairs an incorrect parsing decision committed by the parsing oracle. For each parsing failure, all candidate transitions are mapped to individual feature vectors through the use of the feature model presented in Section 6.5.1.

The training data was acquired by parsing articles from WikiWoods, section 4000 and onwards. We required two different models, corresponding to the two parsing modes to which backtracking apply, namely CFG approximation mode and HPSG unification mode. Due to a higher number of parsing failures in unification mode, more training instances were collected in this mode when using an equal amount of training data.

After evaluating various feature models combined with a variable amount of training data, we decided to use a training model consisting of 150,000 sentences for the CFG model, and 50,000 for the unification model. In total approximately 76,000 and 91,000 gold standard backtracking events, and 1,100,000-1,400,000 features were collected for the unification and CFG models, respectively.

6.6 Baseline Strategy

Following the 'Optimistic' constraints, e.g. never backtrack farther than the last backtracking decision, we designed an alternative backtracking strategy, loosely inspired by Ninomiya et al. (2011). Rather than ranking each candidate transition by a ranking model, we select the candidate transition ($c_c.t_c$) with the highest $PP(c_c)$ score. In Section 6.5.1 we described this as

the probability proportion of the second-highest scoring probability $P(c_c.t2)$ over the sum of the probabilities $P(c_c.t2)$ and $P(c_c.t_c)$: $\frac{P(c_c.t2)}{P(c_c.t2)+P(c_c.t_c)}$

In Section 6.4.3 we discussed the potential benefits of using a ranker to determine the backtracking points, compared to a method that only considered the local context of each transition. This alternative baseline approach allows us to evaluate this hypothesis. By keeping the remaining backtracking constraints identical, and using a realistic metric to determine a backtracking point for a local transition context, this allows us to gauge the potential benefits and disadvantages of using a ranking model to navigate the decisions of the backtracker, compared to a strategy that only considers the local context of each transition candidate $c_c.t_c$. We will present the accuracy for this baseline approach, together with the default ranking-based method in Section 6.7 below.

6.7 Backtracking Accuracy

In the following, we will present and evaluate the precision for the optimistic backtracker in isolation when used as an overlay to CuteForce. We will compare the findings for the ranking-based approach (see Section 6.5), and the baseline strategy (Section 6.6). The actual parsing results will be presented in Chapter 7 – in this section we will evaluate the backtracker's ability to repair an incorrect parsing decision which subsequently lead to a parsing failure. When the first parsing failure in the parse of a sentence occurs in configuration c_f, each transition in the sequence $c_f.T$ are candidate backtracking points – we know that in the sequence $c_f.T$ there is exactly one transition that diverges from the gold standard search path, and the task of the backtracking ranker and the baseline method is to identify this transition. To reduce noise, we have only used gold standard supertags in the evaluation of the test data. We here present the results for the development data set *(we12)*, as well as the results for the in-domain test set *(ws13)*.

Tables 6.3 and Table 6.4 present the statistics regarding the scope of the backtracking problem, and the accuracy for the ranking-based strategy (Table 6.3) and the alternative baseline approach (Table 6.4). *Mode* refers to the parsing mode for the CuteForce Parser. *BT* is the proportion of sentences from the data set that reaches a parsing failure, and are hence subject for backtracking.[3] *BT Cand* refers to the average number of backtracking candidates which is considered, this is the average of the total number (f) of transitions that were executed before parsing failed in c_f. Correspondingly, *Acc.* provides the accuracy for the backtracker, i.e. the proportion of correct classification choices where the first incorrect parser decision is located. *Corr t2* is the proportion of times where the parsing oracle's second-highest ranked parse transition ($c_e.t2$) turned out to be the correct parse transition

[3]This is the inverse of the coverage when parsing in deterministic mode.

when the incorrect parsing transition $c_e.T_e$ was identified. *Corr BT* is the product of *Acc.* and *Corr t2*, hence the estimate of the proportion of parsing failures where the backtracker both identifies the configuration in which an incorrect parsing decision was taken (c_e), and replaces this with the correct transition for the configuration c_e, returning the parser to the gold standard path.

Data	Mode	BT	BT Cand	Acc.	Corr t2	Corr BT
ws12	CFG	23.1	56.5	25.2	81.1	20.4
	Unify	47.4	52.7	24.9	39.2	9.8
ws13	CFG	19.1	59.3	24.0	77.8	18.7
	Unify	43.3	55.8	21.5	38.4	8.3

Table 6.3: Backtrack statistics for the ranking-based backtracker

Data	Mode	BT	BT Cand	Acc.	Corr t2	Corr BT
ws12	CFG	23.1	56.5	18.0	85.1	15.3
	Unify	47.4	52.7	18.2	37.8	6.9
ws13	CFG	19.1	59.3	20.0	83.4	16.7
	Unify	43.3	55.8	19.1	36.3	6.7

Table 6.4: Backtrack statistics for the baseline backtracking approach.

From Table 6.3 and 6.4, we immediately see that the magnitude of the classification task is large. The average number of candidate transitions upon the first parsing failure is in the range from 52-59 – randomly selecting a transition would have an estimated success rate of less than 2%.

The ranking-based backtracking classifier has an accuracy ranging from 21.5-25.2%, and the baseline strategy is doing worse in all configurations, with an accuracy ranging from 18-20%. It is also interesting to note that there are large parsing mode asymmetries in the parsing oracle's prediction of the second-highest ranked parsing decision when the highest ranked transition did not turn out to be correct. When the parsing oracle makes a mistake in CFG mode, the second-highest ranked transition candidate was the correct transition in about 80% of the instances. For unification mode, this percentage is under 40%. Our best explanation to this is that the unification filter allows fewer 'obvious' errors to be made. However, we have not carried out a further study to try to uncover the nature of this effect.

For the overall accuracy, *Corr BT*, we find the ranking-based approach to be more accurate than the baseline strategy in all of the four test sets. $c_e.t2$, the transition that will replace $c_e.T_e$ when c_e is selected as the backtracking point, is not defined by the backtracker, but the parsing oracle. Repairing an incorrect parse decision is clearly harder in unification mode for both backtracking strategies.

The parsing results when applying Optimistic Backtracking are presented in the *Result and Evaluation* chapter, Section 7.6.

6.8 Conclusion

In this chapter we have presented a backtracking overlay for a deterministic transition-based parsing algorithm. The method, *Optimistic Backtracking*, is in itself heavily constrained to optimize parsing performance and for ease of implementation, and provides a simple and highly efficient strategy for backtracking. Although it has been designed to be implemented in the Cute-Force platform, we assume that this backtracking approach can be applied to other deterministic parsers with relative ease.

A novelty in our backtracking approach is the use of ranking to determine the backtracking point in the sequence of transitions that lie prior to a parsing failure. Applying ranking allows us to combine global and local syntactic and semantic information for each given candidate transition. To our knowledge, this is the first backtracking method that makes use of global information when reverting to a prior state. To evaluate the potential benefits of using a ranking-based approach, we described a baseline 'traditional' backtracking approach which only considers the local probability distribution for the list of transitions for each applicable parser configuration.

Further, we have presented the 'Optimistic' constraints (see Section 6.4) that ensures a dynamic restriction to the number of backtracking attempts that is allowed for each sentence. By honoring the premise of always assuming that the *last* backtrack was correct, *Optimistic Backtracking* can be implemented with only minor adjustments and additional complexity to a deterministic transition-based parser.

In Section 6.7 the accuracies of the backtracking approaches are presented. By comparing the ranking-based approach to a our baseline strategy, we see that the ranking-based method reaches a higher level of accuracy in all of the test configurations. By this we have shown that using ranking for determining the backtracking points for a backtracker is a viable strategy which may increase the backtracker's accuracy compared to traditional methods that only consider local transition probabilities.

Chapter 7

Results and Evaluation

In this chapter we present and discuss the empirical results and overall parsing performance of CuteForce. The parser makes use of the resources which have been developed over the course of this project, specifically the WikiWoods parsed corpus and WeScience Treebank (introduced in Chapter 3) and the supertagger, which provides the input buffer (including preterminals) for the parser (presented in Chapter 4). In the development of CuteForce, many setups have been considered, tested and evaluated. These were designed to explore the nature of the deterministic and near-deterministic transition-based parsing algorithm (presented in Chapters 5 and 6, respectively), the machine learning model, the backtracker and other aspects of the parsing platform.

We will start this chapter (Section 7.1) with a presentation of the syntactic and semantic evaluation metrics that will be applied to all parsing analyses produced by CuteForce. In Section 7.1.6 we present the various parameters and configurations that CuteForce supports, and which will be subject for evaluation. In Section 7.2 we will introduce other parsers that are relevant to our project in terms of methodology and grammatical framework. Most prominently, we will discuss the transition-based HPSG parser developed by Ninomiya et al. (2011, 2009), and elaborate on methodological and technical differences between this parsing platform and CuteForce.

Section 7.3 discusses the effect the training data has on the accuracy of the parsing oracle. Using variable sources and sizes of training data, we plot the accuracy of the parsing oracle on the development data sets, and decide based on these findings what will constitute the training data that will be used in the training of the default parsing oracle for CuteForce.

Sections 7.4 and 7.5 presents the evaluation for the deterministic parsing algorithm in the various parsing modes supported by CuteForce, and Section 7.6 presents an evaluation of CuteForce in near-deterministic mode, allowing a limited amount of backtracking. In this section, a ranking-based backtracking approach, *Optimistic Backtracking*, is used and evaluated to-

gether with an alternative, more traditional backtracking approach. These backtracking strategies were presented in Chapter 6.

In Section 7.7 the parsing results are compared with two other parsers of different technological premises. By comparing our parsing results to other ERG parsers, we can assess how well CuteForce performs in this landscape, and which trade-offs we see in terms of parsing accuracy, complexity and efficiency between the various parsers.

7.1 Parser Evaluation

Most intrinsic parser evaluations are based on treebanks. Hence, they typically assume the somewhat crude simplification that each utterance has one single correct interpretation and analysis, namely the analysis agreed upon by the annotators. Despite its weaknesses, this setup will also be the basis for our evaluation. Parser evaluation is in itself a subfield within NLP, and a number of evaluation metrics have been proposed. Workshops and conference tracks addressing parser evaluation have been a recurring theme for at least the last 20 years. As individual grammatical frameworks have different requirements, a number of evaluation schemes have emerged. The *PARSEVAL* metric (Black et al., 1991) has been a dominating metric for phrase structure grammars. Further, for dependency grammar we find evaluation schemes targeted towards argument structures, used by prominent parsers such as MaltParser (Nivre et al., 2007) and MSTParser (McDonald et al., 2005).

However, we have not seen the same consolidation of evaluation schemes for deeper, semantic grammars – an issue which is addressed in Dridan and Oepen (2011):

> The current range of parser evaluation metrics all evaluate the syntactic quality of parser output, which makes them unsuitable to evaluate parsers which aim to output semantic analysis. (Dridan and Oepen, 2011, p. 228)

Acknowledging the need for a more semantically oriented evaluation scheme, Dridan and Oepen (2011) proposed a framework for evaluation of semantic analyses, dubbed *Elementary Dependency Match* (EDM). For the semantic evaluation of CuteForce, we use this framework, which will be presented in Section 7.1.2. Additionally, the syntatic representation of the CuteForce derivations will be scored by PARSEVAL, using the bracket scoring program EVALB[1] (see Section 7.1.1).

[1] `http://nlp.cs.nyu.edu/evalb/`

7.1.1 Syntactic Evaluation with PARSEVAL

The PARSEVAL metric (Black et al., 1991) is a measure of the syntactic overlap between the parsed derivation and the gold standard. Applying the conventional tree representation of an HPSG derivation as depicted in Figure 7.2, we use EVALB to provide the scores relating to *Matched Bracket*, *Sentence Accuracy* and *Tag Accuracy*. EVALB uses a PARSEVAL evaluation configuration identical to the algorithm which is presented in Collins (1997), and for the evaluation of CuteForce, we will apply this metric to score the overlap between the parsed CuteForce derivations and the gold standard analysis.

In the syntactic evaluation of CuteForce, we will use the following metrics, all of which are computed by EVALB.

Matched Bracket

In this metric, the *constituents* of the parsed HPSG analysis are matched against the gold standard derivation. A constituent is a conventional CFG rule instantiation, i.e. the projection of a mother rule and its daughters. For the example sentence *He persuaded Kim to leave*, the tree derivation in Figure 7.2 contains nine constituents that will be considered by PARSEVAL.

Consitutent	Start	End
sb-hd_mc_c	0	5
hdn_bnp-qnt_c	0	1
hd-cmp_u_c	1	5
hd-cmp_u_c	1	3
v_pst_olr	1	2
hdn_bnp_pn_c	2	3
hd-cmp_u_c	3	5
hd_optcmp_c	4	5
w_period_plr	4	5

Table 7.1: Constituents from the tree derivation in Figure 7.2.

The *Start* and *End* indeces refer to the positioning of the tokens that the constituents span. We notice that preterminals are not considered in Table 7.1. They are however scored in the *Tag Accuracy* metric. Each matching bracket is counted as a *correct constituent*. The definition of 'correct' is elaborated in (Collins, 1997):

> For a constituent to be 'correct' it must span the same set of words (ignoring punctuation, i.e. all tokens tagged as commas, colons or quotes) and have the same label as constituent in the treebank parse. (Collins, 1997, p. 22)

Following this definition of a correct constituent, the matched bracket recall and precision is hence computed as:

- Bracketing Recall = $\frac{\#CC}{\#GC}$
- Bracketing Precision = $\frac{\#CC}{\#PC}$

where $\#CC$ is the total number of correct constituents in the parsed derivations, #GC is the total number of constituents in the gold standard and #PC is the total number of constituents in the parsed derivations. In the syntactic evaluation, we will provide the harmonic mean between the precision (PR) and recall (RE), namely the F-measure:

- F-measure = $2 * \frac{PR*RE}{PR+RE}$

These three metrics, *Recall*, *Precision* and *F-measure*, will be presented in the syntactic evaluation.

Sentence Accuracy

The *sentence accuracy* is the proportion of sentences where all constituents from the gold standard derivation have found a match in the parsed derivation. In other words, this means that the bracketing F-measure for the derivation is 100%.

Tag Accuracy

The *tag accuracy* represents the supertag accuracy for the preterminals. In our case, this concerns the supertags, and we will evaluate the parser using gold standard supertags (in this event, the tag accuracy is 100%), or tagged preterminals, using the supertagger presented in Chapter 4.

7.1.2 Semantic Evaluation

As opposed to the syntactic evaluation using PARSEVAL, the semantic evaluation we carry out will consider the underlying semantics that is expressed through the HPSG derivations produced by CuteForce. Similar to most initiatives within the DELPH-IN community, the default semantic representation is Minimal Recursive Semantics (MRS; Copestake et al., 2005), and we will in the following discuss the aspects of MRS which are crucial for our evaluation. We refer to Copestake et al. (2005) for a more comprehensive introduction to MRS.

Minimal Recursion Semantics

MRS is a flat semantic formalism that represents semantics by the use of *elementary predications* (EP) and underspecified scopal constraints. An

EP will either relate to the tokens of the sentence, or to a grammatical construction. In Figure 7.1 we see the MRS for the sentence *He persuaded Kim to leave.* (MRS examples are derived from Dridan and Oepen, 2011.) Correspondingly, the ERG tree derivation for the same sentence is found in Figure 7.2.[2]

$\langle h_1,$
h_3:pron<0:2> (ARG0 x_4{PERS *3*, NUM *sg*, GEND *m*, PRONTYPE *std_pron*}),
h_5:pronoun_q<0:2> (ARG0 x_4, RSTR h_6, BODY h_7),
h_8:_persuade_v_of<3:12> (ARG0 e_2{SF *prop*, TENSE *past*, MOOD *indicative*}, ARG1 x_4, ARG2 x_{10}, ARG3 h_9),
h_{11}:proper_q<13:16> (ARG0 x_{10}{PERS *3*, NUM *sg*}, RSTR h_{12}, BODY h_{13}),
h_{14}:named<13:16> (ARG0 x_{10}, CARG *Kim*),
h_{15}:_leave_v_1<20:26> (ARG0 e_{16}{SF *prop-or-ques*, TENSE *untensed*, MOOD *indicative*}, ARG1 x_{10}, ARG2 p_{17})
$\{ h_{12} =_q h_{14},\ h_9 =_q h_{15},\ h_6 =_q h_3 \} \rangle$

Figure 7.1: MRS representation of the sentence *He persuaded Kim to leave.*

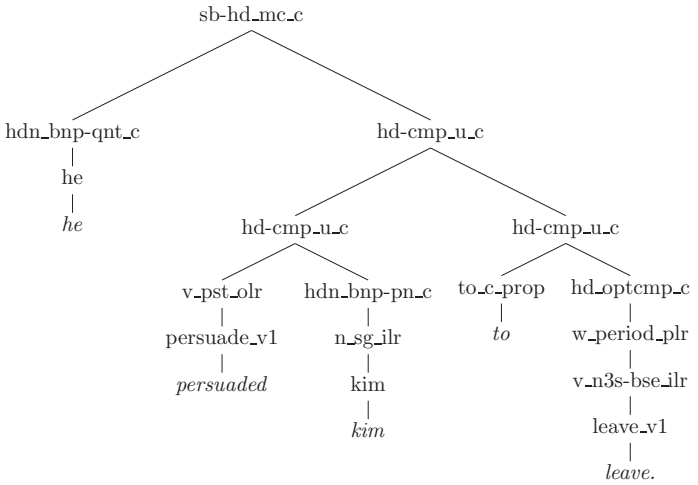

Figure 7.2: ERG tree derivation of the sentence *He persuaded Kim to leave.*

In the MRS in Figure 7.1 we find six EPs, labeled *h3*, *h5*, *h8*, *h11*, *h14* and *h15*, four of which refer to lexemes in the sentence. In addition to the label, each EP consists of a relation name (e.g. *_leave_v_1* in Figure 7.1) and argument relations denoted *ARGx*, where the *ARG1*, *ARG2*, and *ARG3* describe predicate-argument relations, and ARG0 is the *distinguished variable*: "For most types of relations, the distinguished variable corresponds to its main index (ARG0 in MRSs), e.g. an event variable for verbal relations,

[2]The *root* category, in this case *root_strict*, is omitted from the derivation because it does not contribute to the semantic representation of the sentence, and is therefore not evaluated.

a referential index for nominals." (Oepen and Lønning, 2006, p. 1252) In the argument structure of *_persuade_v_of*, we find *ARG1* referring to (the semantic representation associated with) *He*, *ARG2* refers to *Kim* and *ARG3* refers to *leave*. Further, properties such as gender and tense are expressed through entity and event variables. This framework facilitates the use of *Elementary Dependency Match* (Dridan and Oepen, 2011) as an evaluation metric, and this will be presented below.

Elementary Dependency Match

Elementary Dependency Match (EDM) is a parser evaluation metric for parsers which produce semantic analyses in the Minimal Recursion Semantics formalism (MRS; Copestake et al., 2005). It is based on *Elementary Dependency Structure* (EDS), which is a 'variable-free' reduction of MRS developed by Oepen and Lønning (2006), where the goal is to "[compare] semantic properties across multiple analyses, e.g. the set of competing parses for a token input, since there is no (straightforward) way of making sure that related pieces of semantics across analyses actually use parallel variables." (Oepen and Lønning, 2006, p. 1252) This reduction has been favored in a number of downstream processing applications, e.g. discriminants for semantics-based treebanking (Oepen and Lønning, 2006), parse selection (Fujita et al., 2007), in addition to the semantic parser evaluation, *EDM*, which we will use for the semantic evaluation of CuteForce. An EDS is a graph, which in its turn can be further reduced to a set of triples. An EDS facilitates a distinction of three evaluation classes which are applied in EDM (Dridan and Oepen, 2011):

1. core functor-argument structure, whether syntactic or semantic
2. predicate information, such as the lemma, word category and sense
3. properties of events and entities, such as tense, number and gender

Figure 7.3 demonstrates how the evaluation classes are determined from the semantic structure. In this representation, each token from the MRS in Figure 7.1 is equipped with triples with its (1) argument roles (EDM_A), (2) name relation (EDM_N) and (3) (EDM_P) event and entity properties, corresponding to the evaluation classes presented above. Additionally, this representation also includes sub-string character span to identify the nodes in the graph (we do not consider the actual sub-string a part of the triple).

An Elementary Dependency Match over the gold standard representation and the parsed representation will compute the overlap of triples for the evaluation metrics, enabling the computation of a recall, precision and F1-score for each of the three classes, or a combined score over all triples depicted in Figure 7.3.

In the evaluation of CuteForce, we will use **class 1** (EDM_A), and **class 3** (EDM_P) in the semantic evaluation. The first category from the evaluation

EDM_A	"He"	<0:2>	ARG0	<0:2>	"He"
	"persuaded"	<3:12>	ARG1	<0:2>	"He"
	"persuaded"	<3:12>	ARG2	<13:16>	"Kim"
	"persuaded"	<3:12>	ARG3	<20:26>	"leave."
	"Kim"	<13:16>	ARG0	<13:16>	"Kim"
	"leave."	<20:26>	ARG1	<13:16>	"Kim"
EDM_N	"He"	<0:2>	NAME	pronoun_q	
	"He"	<0:2>	NAME	pron	
	"persuaded"	<3:12>	NAME	_persuade_v_of	
	"Kim"	<13:16>	NAME	proper_q	
	"Kim"	<13:16>	NAME	named	
	"leave."	<20:26>	NAME	_leave_v_1	
EDM_P	"He"	<0:2>	GEND	*m*	
	"He"	<0:2>	NUM	*sg*	
	"He"	<0:2>	PERS	*3*	
	"He"	<0:2>	PRONTYPE	*std_pron*	
	"persuaded"	<3:12>	MOOD	*indicative*	
	"persuaded"	<3:12>	SF	*prop*	
	"persuaded"	<3:12>	TENSE	*past*	
	"Kim"	<13:16>	NUM	*sg*	
	"Kim"	<13:16>	PERS	*3*	
	"leave."	<20:26>	MOOD	*indicative*	
	"leave."	<20:26>	SF	*prop-or-ques*	
	"leave."	<20:26>	TENSE	*untensed*	

Figure 7.3: Gold triples for *He persuaded Kim to leave.*

metrics, EDM_A, is typically used with dependency grammar, evaluating head-dependent relations. This metric offers a cross-platform guideline to the accuracy of the dependency relations produced by the parser. In the triple representation in Figure 7.3, we see a head-dependent relation from *persuaded* to its arguments *He* (ARG1), *Kim* (ARG2) and *leave* (ARG3).

The third class, labeled EDM_P, refers to attributes such as TENSE, NUMBER and PERSON. In Figure 7.3 we find the triple ⟨ "persuaded", TENSE, *past*⟩, defining the verb "persuaded" to be in past tense. Although the same information is captured in the ERG tree derivation in Figure 7.2, there may not always be a one-to-one relation between a lexical rule in the tree representation, and an EDM_P triple. In the tree derivation in Figure 7.2, the tense of the verb is expressed through the lexical rule *v_pst_olr*.

The second class evaluates relations through the predicate names. This is specific to the preterminals, and has been omitted because the preterminals assumed by CuteForce, the lexical types, are not equivalent to the preterminals applied in the treebanks, namely lexical entries. The alternative preterminal granularities were discussed in Section 4.1.3.

Robust Unification

When parsing in CFG- or unrestricted parsing mode, the transitions, and subsequent parse derivations, are not verified against grammatical constraints, hence it is not guaranteed that the final parse derivation represents a consistent, unifiable HPSG analysis. CuteForce has this feature in common with Jigsaw (introduced in Section 2.5.1). For this PCFG parser, Zhang et al. (2012) suggest *Robust Unification* in order to resort inconsistencies in an HPSG structures from a parse derivation that potentially violates grammatical constraints:

> Naive post-processing of a CFG derivation in the HPSG universe will frequently give rise to unification failures. To work around these issues, we couple our PCFG parser with a robust procedure for meaning composition, grounded in default unification, that seeks to maximize the amount of semantic information available from the robust context-free analyses. (Zhang et al., 2012, p. 3)

An inconsistent HPSG derivation can imply conflicting information, such as incompatible daughter categories for some syntactic constructions or inconsistent number values for two semantic variables that should be unified. Zhang et al. (2012) propose to overwrite unification conflicts by deleting or overwriting the conflicting inconsistencies, parallel to so-called *default unification*: "In doing so, our abstract goal will be to maximize the quality of semantic information, while keeping to the deterministic nature of the post-parsing reconstruction of PCFG derivations."(Zhang et al., 2012, p. 17) A similar strategy is also pursued for the HPSG parser developed by Ninomiya et al. (2011), where default unification in the context of transition-based HPSG parsing is further discussed.

We have applied the framework for robust meaning composition developed by Zhang et al. (2012) in order to extract well-formed MRSs from parse derivations produced by CuteForce, hence facilitating evaluation using Elementary Dependency Match for all CuteForce analyses. The running time of this post-processing stage is not accounted for in the parsing times that are reported for CuteForce and Jigsaw (Section 7.4-7.7). In its current state, the robust unification framework is not optimized for efficiency, and our evaluation indicated that the average processing times were in the range of 75-150 milliseconds per derivation.

7.1.3 Significance Testing

We have used Dan Bikel's *Randomized Parsing Evaluation Comparator*[3] to perform significance testing using *stratified shuffling* (Cohen, 1995) over

[3] `http://www.cis.upenn.edu/~dbikel/software.html#comparator`

syntactic analyses, using the EVALB output. This test has been applied for a few CuteForce configurations where we have found that the outcome of our parsing results have been inconclusive with respect to whether two alternative configurations produce significantly different parsing accuracy. We assume a p-value of below 0.05 in order to regard two alternative parsing results to be significantly different.

We have however not conducted pairwise significance testing towards the alternative ERG parsers that we present in Section 7.7. For these parsers (Jigsaw and PET HPSG Parser), we only had available the PARSEVAL scores for the overall test sets, and not an EVALB report that outputs the evaluation on a sentence-level, which would be required to perform stratified shuffling.

7.1.4 Parsing Statistics

In addition to the syntactic and semantic evaluation presented in Sections 7.1.2 and 7.1.1, we present statistics which do not specifically relate to the individual parsing results which are produced. In the following, we will introduce the eight categories for which parser statistics are presented. Their abbreviations in the tables are parenthesized.

Coverage

The *Coverage* (COV) refers to the proportion of sentences in the test data that successfully received a parse. Hence, the inverse of this metric reports the proportion of sentences where parsing failed, and CuteForce was not able to provide a parse analysis.

Consistency

This metric presents the proportion of sentences in the test data that yields a parse derivation which is *consistent* (CON) with the unification-based grammar we are parsing in, namely ERG. Hence, a *consistent* parse derivation does not violate any grammatical constraints within the framework, which means that the final derivation represents a unifiable HPSG sign. The inverse of this metric represents the proportion of sentences where we had to make use of robust unification in order to resort inconsistencies in the HPSG analysis.

Average Processing Times (All Sentences)

The Average Parsing Times (PT) are the average number of milliseconds used for processing each sentence in the test set. This excludes pre-processing by the supertagger, which took approximately 0.05 milliseconds per token (see Tables 4.5 and 4.6), and post-processing using robust unification (see

Section 7.1.2). The experiments are carried out on an Intel Xeon X7550 Server with 2.0 GHz CPUs, and the entire CuteForce platform is developed in Java (version 1.6). Since we did not have a dedicated node, a variable amount of load on the server may to some degree have affected the parsing times.

Average Processing Times (Parsed Subset)

This average (PT-P) is computed over the processing times (excluding pre/post-processing) for the sentences which were successfully parsed.

Average Processing Times (Failed Subset)

This average (PT-F) is computed over the processing times (excluding pre-processing) for the sentences which ended in parsing failure.

Average Parsed Sentence Length

This metric (SL-P) presents the average number of tokens for the subset of the test data that was successfully parsed. If coverage is 100%, the average sentence length would be identical to the average length for the test set.

Average Failed Sentence Length

This metric (SL-F) presents the average number of tokens for sentences where parsing failed.

Tagger Accuracy

The *Tagger Accuracy* (TA) is the total number of correct preterminals in the subset of sentences that CuteForce parsed for a given parsing mode, divided by the total number of preterminals in this subset. The metric is presented among the parser statistics, since it is not related to the parser per se, but is derived from the input buffer, where the preterminals are tagged by the supertagger (as presented in Chapter 4) in a pre-processing stage. This metric is of course not applicable when using gold standard preterminals. We will see that when coverage is 100%, the tag accuracy is identical to the supertag accuracy presented in Table 4.5. However, in configurations with lower coverage, the subset of parsed sentences is altered, and the tag accuracy for this specific subset is presented here.

7.1.5 Error Analysis

For each configuration we present an error analysis relating to the parsing mistakes committed by CuteForce. Each gold standard derivation is produced through one explicit sequence of transitions. In the error analysis we

will present statistics for the transitions produced by CuteForce, and how this sequence of transitions overlapped and deviated from the gold standard transitions. This statistic is based on the sequence of transitions leading up to a successful parse derivation that is in compliance with the gold standard, or up to the point where the parser executes a transition that diverges for the correct search path. The statistic allows us to evaluate the nature of the errors produced by CuteForce, and to examine strategies that may accommodate for these mistakes. During the development of CuteForce, this insight was of particular importance, particularly in the design of the oracle's feature model (the feature model was presented in Section 5.3.6).

For each CuteForce configuration we will present transition statistics when parsing with tagged preterminals on *ws13*.

7.1.6 Evaluated CuteForce Setups in This Thesis

In Chapter 5 we gave a comprehensive presentation of the CuteForce parsing platform that enables deterministic, transition-based HPSG parsing. Although a number of strategies and alternative approaches have been evaluated during the development of the parser, we have decided to fix most of the parser-related configurations.

Especially concerning the machine learning model one could imagine countless variations of adjustments that would have triggered different behaviors by the oracle (e.g. the removal or inclusion of individual features in the feature model presented in Section 5.3.6). In Section 7.3 we present how the accuracy of the oracle gradually improves as more data is added to the pool of training sentences. We only present the findings when using the deterministic, unrestricted parsing mode with gold standard preterminals. Even though CuteForce can operate in a large number of configurations, as will be presented in Figure 7.4, the objective of the oracle will always remain the same: *Predict the correct next transition given the current parser configuration.* Due to this underlying premise which applies to all setups of CuteForce, as depicted in Figure 7.4, we have opted to use the same machine learning model in the evaluation of all CuteForce configurations that use the single-stage transition system, as we found it implausible that a machine learning model alternative to the default setup would yield significantly improved parsing results.

Figure 7.4 presents the alternative configurations for CuteForce as a tree of relevant dimensions of parameterization. The highest level represents the search space that is explored by CuteForce, and the restrictions thereto: The deterministic parsing algorithm (as presented in Chapter 5), and the near-deterministic configuration that facilitates moderate backtracking (presented in Chapter 6).

As we discussed in Section 5.3.2, we evaluated a two-stage transition system for the deterministic CuteForce configuration, together with the single-

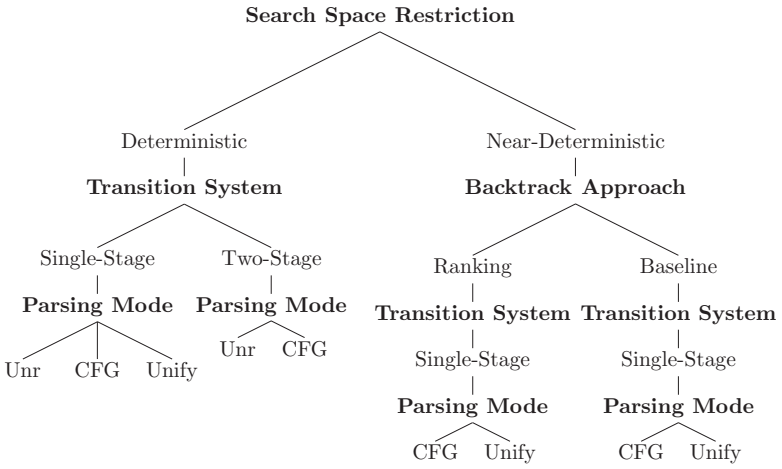

Figure 7.4: The configurable parameters of CuteForce.

stage oracle setup which became the default transition scheme. Only limited support is offered for the two-stage classification setup, and in Figure 7.4 we see that this operation is only evaluated in the deterministic configuration for the unrestricted and CFG approximation parsing modes. The two-stage transition system required a different oracle setup, and allowed for the use of more training data. This is further discussed in Section 7.5.

The *Parsing Mode* dimension in Figure 7.4 refers to the various parsing modes, as presented in Section 5.2.5. We remember that the deterministic CuteForce parser can operate in *unrestricted mode* (Unr), *CFG approximation mode* (CFG) and *HPSG unification mode* (Unify). All three options are however only available for the deterministic configuration using the default single-stage transition system. Whereas the unification mode is unsupported for the two-stage transition system, the unrestricted mode is not meaningful when parsing in near-deterministic mode. For the latter part, this is because the backtracker utilized in the near-deterministic configuration will only be launched when a parsing failure occurs, and a parsing failure will never happen when parsing in unrestricted mode (the concept of parsing failure in near-deterministic parsing is elaborated in Section 6.4.4). Further, for near-deterministic parsing, we have two different backtracking approaches, namely *Ranking* (presented in Section 6.3) and *Baseline* (see Section 6.6).

Each of the nine leaves in Figure 7.4 represents an individual configuration of CuteForce. Each of these configurations is evaluated on a test batch consisting of an in-domain and out-of-domain data set. Figure 7.5 represents the evaluations that are conducted for each of the CuteForce configurations

in Figure 7.4.

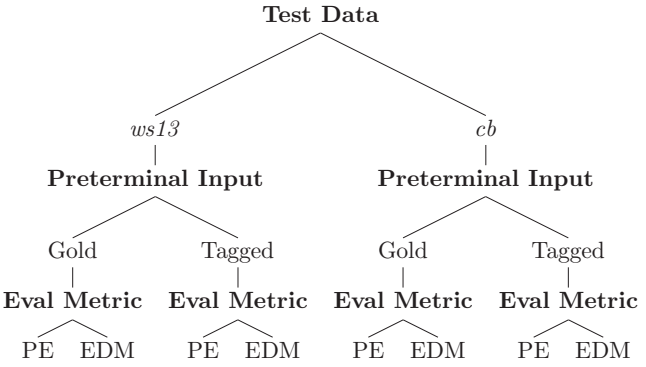

Figure 7.5: The evaluation that is performed for every CuteForce configuration.

As we see in Figure 7.5, the evaluation consists of the test sets *WeScience, Section 13 (ws13)*, which is in-domain for the parser, and *The Cathedral and the Bazaar (cb)*, which is our out-of-domain test data (the data sets are presented in Section 3.5). Each of the test sets will be evaluated using gold standard preterminals, or tagged preterminals using the supertagger we presented in Chapter 4. Finally, *Eval Metric* refers to the syntactic and semantic evaluation that is conducted, as presented in Section 7.1. We see that all test sets will be evaluated both through a syntactic evaluation (PARSEVAL), and a semantic evaluation, using Elementary Dependency Match (EDM).

Totally we see that for each of the nine CuteForce configurations in Figure 7.4, eight different evaluations are performed (see Figure 7.5). Further, we will in Section 7.7 present parsing scores for two related parsers on the same test data, and discuss the performance of CuteForce compared to the related parsers in this field.

7.2 Evaluation Against Alternative HPSG Parsers

To our knowledge, the parser presented in Ninomiya et al. (2011, 2009) is the only effort in transition-based deterministic and near-deterministic parsing for HPSG, alongside with CuteForce. However, a head-to-head comparison of the parse analyses produced by the two parsers is not possible due to formal and technical differences in the underlying grammars. Instead of directly comparing the two parsers, we can however see how they relate to traditional non-deterministic parsers for the same grammatical frame-

work. Ninomiya et al. (2011) presented an extensive evaluation of their (near-)deterministic parser, comparing the parsing performance to other non-deterministic parsers for the Enju framework. In the same manner, we will in Section 7.7 present the parsing performance for the PET HPSG Parser (see Section 2.5.3), which is the 'standard' HPSG parser for the DELPH-IN consortium, together with Jigsaw (Zhang and Krieger, 2011), a PCFG parser trained on WeScience and WikiWoods (see Section 2.5.1), and discuss how these parsers relate to CuteForce in terms of parsing accuracy and efficiency.

In the following, we will however discuss how CuteForce and the Enju parser presented in Ninomiya et al. (2011) relate to one another from a technical and methodological perspective, although we were unable to directly compare the parsing accuracy for the two parsing platforms.

7.2.1 Comparison of CuteForce and Ninomiya et al. (2011)

CuteForce and the transition-based HPSG parser presented in Ninomiya et al. (2011, 2009) have been developed partially simultaneously, and the Enju parser represents a methodology that shares many insights with CuteForce. In the following we present some important aspects with the two parsers, and discuss how they are handled for the two parsing platforms.

Underlying Grammar

While the Enju HPSG (Miyao et al., 2004, see Section 2.5.4) used by Ninomiya et al. (2011) is induced from the Penn Treebank (PTB; Marcus et al., 1993), the ERG (see Section 3.2) is a handcrafted grammar. Where the granularity of an induced grammar like Enju is largely determined by information available in the treebank, the ERG can include more fine-grained and richer analyses, for example with respect to subcategorization patterns (e.g. including relational nouns and adjectives), multi-word expressions and other subtle linguistic properties that cannot easily be extracted from the PTB with heuristics alone.

Even if the parser proposed by Ninomiya et al. (2011) had been publicly available, adapting it to ERG would not be trivial, because the typed feature structure logic assumed in the ERG is formally richer than Enju's. Technical difficulties associated with different linguistic expressiveness are also discussed in Miyao et al. (2000, p. 50), where the LiLFeS abstract machine, which is the underlying programming language applied by the Enju grammar, is contrasted to the ERG. Nor have we attempted to adapt CuteForce to the Enju grammar, since the DELPH-IN grammars have been the target for our research in this project.

Unification

The HPSG parser developed by Ninomiya et al. (2011) can operate in two different unification modes: *Normal* and *default*. Using normal unification, the parser will only execute transitions that imply a valid unification, leaving the parser exposed to parsing failures. By using default unification the parser will on the other hand overwrite inconsistencies in the feature structure if the parser reaches a configuration where there are no further available transitions that implies a valid unification. Hence the parser achieves robust processing, whilst maximizing the amount of information in the feature structures (Ninomiya et al., 2011, p. 338).

While CuteForce facilitates a parsing mode that ensures that no transitions violate a unification (and thus a grammatical) constraint, it may also operate in another two parsing modes (unrestricted and CFG approximation mode) which omit unification during parsing. By not applying unification as a constraint check, the search space for the parsing oracle is substantially larger, hence placing greater demands on the precision of the classifier that ranks the available transitions. Leaving out unification during parsing, CuteForce can achieve very fast parsing times. Robust unification can optionally be applied in a post-processing stage, thereby ensuring a 100% coverage whilst maximizing the semantic information produced by the parser.

Data Structure

Ninomiya et al. (2011) assume a data structure where the lexical entries and the partial parse trees during parsing are represented by typed feature structures (Carpenter, 1992). CuteForce on the other hand represents the full and partial HPSG derivation as a more conventional phrase structure tree. When CuteForce does not apply unification during parsing, the HPSG parsing would resemble CFG parsing, as described in Sagae and Lavie (2005), although the feature model of CuteForce is augmented with HPSG specific information such as head-relations. The use of a pure and simple phrase structure representation entails that we can delay resolving potential unification violations in the partially built HPSG derivation to a post-processing stage in a straightforward manner, since we are not explicitly manipulating typed feature structures during parsing.

Further, the alternative representation of the (partial) HPSG analyses for the two parsers could have implications in areas such as feature access, but we have not conducted a study to evaluate this issue.

Preprocessing

The parser described in Ninomiya et al. (2011) assumes an input buffer of pairs, consisting of the word item and a part-of-speech tag. The preprocessing stage thereby entails tokenizing and assigning part-of-speech tags

to each token. The parsers 'shift' operation pops a pair from the input buffer, and selects a lexical entry among the candidate lexical entries for the item. Hence, unlike CuteForce, the shift-operation of Ninomiya et al. (2011)'s parser means that supertagging is effectively done during parsing, and not in a preprocessing stage:

> The features for selecting shift actions are the features used in the supertagger (Ninomiya et al., 2007) and the features used in Sagae and Lavie's (2005) parser. Our shift-reduce parsers can be regarded as an extension of the supertagger. (Ninomiya et al., 2011, p. 345)

By delaying the assignment of lexical entries to the parsing stage, Ninomiya et al. (2011) may reduce the probability of propagating mistakes in the preprocessing stage onto the parsing. However, given that part-of-speech tags are assigned prior to parsing, the decisions made during preprocessing will affect the outcome of the parsing. Overall, both parsers rely on a preprocessing stage, but CuteForce will make more fine-grained decisions during preprocessing, compared to the parser in Ninomiya et al. (2011).

Near-Deterministic Strategies

Both parsers facilitate deterministic parsing by applying greedy best-first search. Further, CuteForce supports two alternative backtracking strategies, namely *baseline* and *ranking-based*. While our baseline backtracking approach is inspired by the backtracking strategy described in Ninomiya et al. (2011), the ranking-based approach applied by CuteForce is to our knowledge the first attempt to use ranking in the determination of the backtracking point.

In addition to the aforementioned backtracking approach, Ninomiya et al. (2011) propose a beam-search strategy that allows the exploration of variable proportions of the search-space. These near-deterministic strategies presented in Ninomiya et al. (2011) were also discussed in Section 2.7.4.

7.3 Training Data

In Section 3.5 we presented the training data that is employed in this thesis, and which constitutes the training and development data used when training the parsing oracle. When assessing the impact that different training sources and sizes of the training data have on the accuracy of the parsing model, we followed the same procedure as in Chapter 4 when we presented the supertagger (Section 4.4). Hence, we evaluated three different pools of training data:

1. WeScience treebank (Section 1-11, 7,636 derivations), augmented with WikiWoods
2. Redwoods treebank (29,817 derivations, including WeScience 1-11), augmented with WikiWoods
3. WikiWoods parsed corpus, from segment 2000 and onwards

While the gold standard data is limited, WikiWoods, with its 47 million utterances, provides us with more than enough 'silver' standard data. Similar to the evaluation for the supertagger, we train and evaluate the accuracy of CuteForce for the three different data sets, and with increasing amounts of training data. This evaluation will be presented in Section 7.3.1 below.

7.3.1 Training Size Variation

The performance of a classifier is determined by a complex interaction of the feature selection, the size and quality of the training data, and the nature of the classification task. In Section 5.3.5 we presented the feature selection that we found to be optimal for CuteForce. Applying this feature model, we trained classifiers (e.g. parsing oracles) starting at 7,636 training derivations (e.g. the total number of derivations in WeScience 1-11), up until we reached the maximum number of dimensions that LibLinear facilitated (this limitation is discussed in Section 5.3.5).

Figures 7.6 and 7.7 present learning curves for the deterministic CuteForce configuration, applying gold standard supertags and parsing in unrestricted mode. Figure 7.6 represents the (partially) in-domain data set which has been used during development of CuteForce, namely *ws12*. Additionally, the sentence accuracy for the *ecpr* data set is found in Figure 7.7. *ecpr* is out-of-domain data, and was not used in the development of the parser (see Section 3.5), and is included here as an additional evaluation metric.

In Figure 7.6 we observe that initial performance differences between the three data sets diminish as an increasing number of WikiWoods derivations are added. Beyond the point of 150,000 training derivations, we were not able to successfully train a LibLinear model using the current feature model. We did evaluate models based on larger amounts of training data by reducing the complexity in the feature model, but this did not yield improved results compared to applying our final feature selection.

For the smallest number of training sentences in Figures 7.6 and 7.7 (7,636 sentences) we observe that the sentence accuracies when training on Redwoods treebank is lower than for the other data sets, and this is more pronounced when testing on *ws12*. The beginning of the Redwoods treebank consists of annotated LOGON data, translated tourist guides for Norway, with a high frequency of Norwegian place names, and we expect that the nature of this domain is having a negative effect on the accuracy of the

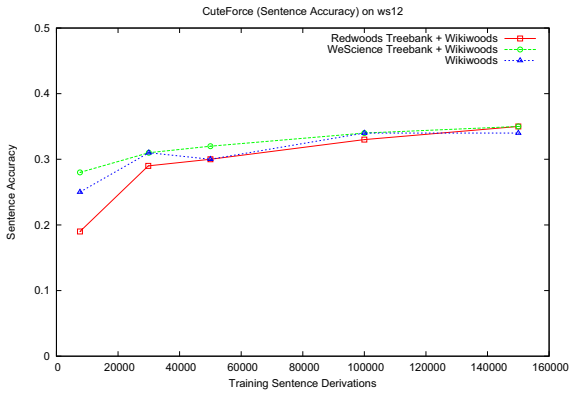

Figure 7.6: The sentence accuracy for the in-domain develoment data set *ws12* increases when adding more training sentences. At 150,000 derivations, there are no substantial difference between the three sources of training data.

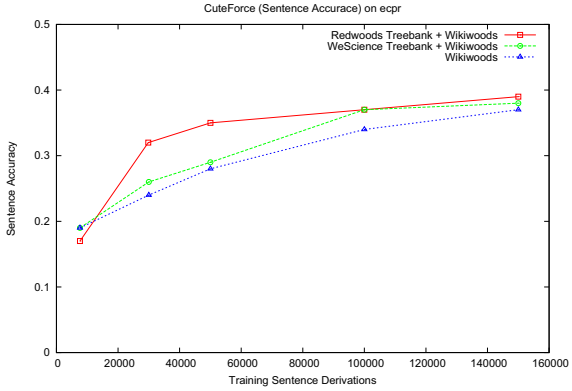

Figure 7.7: The sentence accuracy for the out-of-domain test set *ecpr* increases when adding more training sentences. At 150,000 derivations, there are only minor difference between the three sources of training data.

parser when the training data is so limited.

In Figure 7.7 we see the same reduction in variation between the training sources as more data is added, effectively making the pools of training data more uniform. More interesting is perhaps the accuracy of the models when the size of the training data is in the lower range. We see that the Redwoods treebank stands out as substantially more accurate when the training data

consists of 30,000 sentences, which is the total number of gold standard derivations in the Redwoods treebank. Beyond this point, the differences diminish as additional amounts of WikiWoods data are added to the pool of training data. Although there are only marginal differences in the accuracies for the three training data sources at 150,000 training derivations, we have opted to use the Redwoods treebank, augmented with WikiWoods data up to 150,000 sentences as our pool of training data for all configurations of CuteForce. Although the learning curves for *ws12* (Figure 7.6) which is in-domain for WikiWoods, did not present any substantial differences between the three pools of training data beyond the point of 100,000 training derivations, Figure 7.7 suggests that the gold standard Redwoods treebank in itself provides more reliable training data for the out-of-domain test data, and this is in itself a strong incentive to include the Redwoods treebank in the training data.

7.4 Deterministic Parsing

The various CuteForce setups which we will evaluate were summarized in Section 7.1.6. In this section, we will present the evaluation for the deterministic branch of the parsing platform, applying the default single-stage classification setup. We divide the evaluation in four parts: The parsing statistics, the syntactic evaluation, the semantic evaluation and an error analysis. The overall conclusion for this setup will be delayed to Section 7.8, where we conclude on the parsing performance of CuteForce in terms of accuracy, robustness, and efficiency, and discuss the trade-offs compared to other ERG parsers.

7.4.1 Parsing Statistics

Using the metrics presented in Section 7.1.4, we will in the following present parsing statistics for the deterministic, single-stage configuration of CuteForce. In Tables 7.2 and 7.3 statistics are presented for eight different metrics, we refer to Section 7.1.4 for further details on these categories.

The parser statistics in Tables 7.2 and 7.3 do not explicitly report on the overlap between the parsed derivations and the gold standard. However, the coverage (COV) is vital to any further evaluation for a given evaluation setup, as it describes the proportion of the test set where a parse analysis was provided. In unrestricted mode, the coverage is, as can be expected 100%, and the average sentence length of the parsed subset is hence identical to the average sentence length of the test set as a whole. The coverage drops distinctly in CFG and unification modes, and these trends become more overt when applying tagged preterminals (see Table 7.3).

From *SL-P* and *SL-F* we can infer that longer sentences are more likely to fail. Hence, the subset of parsed sentences has a distinctly shorter average

Data	Mode	COV	CON	PT	PT-P	PT-F	SL-P	SL-F	TA
ws13	Unr	100	51.7	13.4	13.4	N/A	14.1	N/A	N/A
ws13	CFG	80.9	51.8	11.7	10.2	18.1	11.9	23.4	N/A
ws13	Unify	56.7	56.7	2.0k	2.0k	2.0k	8.5	21.4	N/A
cb	Unr	100	31.1	16.6	16.6	N/A	18.4	N/A	N/A
cb	CFG	69.6	31.1	14.6	13.4	17.1	16.2	23.3	N/A
cb	Unify	35.7	35.7	2.1k	2.0k	2.1k	12.2	18.4	N/A

Table 7.2: Parser Statistics for the deterministic CuteForce configuration using gold standard preterminals for the test sets *ws13* and *cb*.

Data	Mode	COV	CON	PT	PT-P	PT-F	SL-P	SL-F	TA
ws13	Unr	100	48.7	12.2	12.2	N/A	14.1	N/A	95.4
ws13	CFG	79.4	48.7	11.6	10.0	17.7	11.8	23.0	95.8
ws13	Unify	52.6	52.6	2.0k	2.0k	2.1k	8.2	20.7	96.9
cb	Unr	100	26.9	14.3	14.3	N/A	18.4	N/A	91.6
cb	CFG	68.0	25.8	14.9	13.7	17.3	16.1	23.1	92.0
cb	Unify	29.2	29.2	2.1k	2.0k	2.1k	11.8	21.1	94.8

Table 7.3: Parser Statistics for the deterministic CuteForce configuration using supertagged preterminals for the test sets *ws13* and *cb*.

sentence length when CFG or (especially) unification mode is used. This will have a direct impact on the syntactic and semantic evaluations, where we see that the precision goes up when these parsing modes are applied, suggesting that the subset of shorter sentence length is easier to parse, and/or that the CFG or unification filter is increasing the accuracy of the parsing oracle. This will be further discussed in the syntactic/semantic evaluations in Section 7.4.2 and 7.4.3.

The *consistency* metric, *CON*, reports unsurprisingly that the unification mode is more likely to produce consistent analyses than the other modes. Since unification mode will filter out all transitions which do not imply a valid unification, all parse derivations produced in unification mode will be consistent. For the CFG and unification modes the scores are quite similar, at least suggesting that the CFG mode is not in any substantial way able to filter out parsing decisions which would imply a breach on a unification constraint.

The overall parsing times, *PT*, are in the range of 11-17 milliseconds per sentence for the unrestricted and CFG modes. When unification is applied, parsing is much slower – approximately 2 seconds per sentence. We expect that a substantial speedup could have been obtained if the unification operation was done natively in CuteForce, and not through an XML-RPC interface to a third-party application, namely LKB (Copestake, 2002). Specifically, the current unification back-end performs non-destructive unification, i.e. it does not take advantage of the deterministic nature of CuteForce, or

of the optimizations for feature structure manipulation discussed in Oepen and Carroll (2002).

The Tagger Accuracy, *TA*, is only relevant for Table 7.3. We see that the accuracies vary somewhat for the different subsets, and it seems that the tag accuracy for the subsets that received a parse in unification mode is higher than for the other subsets. There are, as we see it, two intuitions for this tendency: First, shorter sentences may have a higher overall tag accuracy than longer sentences, and second, a higher tag accuracy in a sentence will in itself reduce the likelihood of a parsing failure.

7.4.2 Syntactic Evaluation

Data	Mode	COV	PR	RE	F1	SA
ws13	Unr	100	85.1	85.3	85.2	41.8
ws13	CFG	80.9	88.5	60.8	72.1	41.9
ws13	Unify	56.7	94.3	33.2	49.1	43.8
cb	Unr	100	82.3	82.5	82.4	20.9
cb	CFG	69.6	85.5	52.6	65.1	20.7
cb	Unify	35.7	91.9	21.9	35.4	22.7

Table 7.4: PARSEVAL scores for *ws13* and *cb* using gold standard preterminals.

Data	Mode	COV	PR	RE	F1	SA
ws13	Unr	100	81.7	81.7	81.7	35.5
ws13	CFG	79.4	84.9	56.9	68.1	35.4
ws13	Unify	52.6	91.2	28.9	43.9	36.1
cb	Unr	100	77.5	77.4	77.4	13.2
cb	CFG	68.0	81.4	48.7	60.9	13.4
cb	Unify	29.2	87.9	16.4	27.7	12.5

Table 7.5: PARSEVAL scores for *ws13* and *cb* using supertagged preterminals.

The syntactic evaluation of CuteForce is a comparison of the tree structures derived from CuteForce, compared to the gold standard derivations. Although presented among the parser statistics (Section 7.4.1), we repeat the coverage for each configuration, because the precision and recall are highly affected by the coverage of the data set.

Tables 7.4 and 7.5 report the PARSEVAL scores for the various parsing modes. For CFG and unification mode we see that the coverage is reduced compared to the unrestricted mode, and only a subset of the test data is parsed. In Tables 7.4 and 7.5 this is reflected by the precision *(PR)* and recall *(RE)* scores. Precision increases for CFG and unification mode, unsurprisingly suggesting that the sentences which failed to parse were sentences

which contributed negatively to the precision of the parser in unrestricted mode. While precision increases when HPSG analyses are derived for a subset of the test data, the recall and F-measure *(F1)* drop distinctly, since the recall is computed over all bracketings in the gold standard.

The sentence accuracy (SA) is relatively stable with respect to the various parsing modes. We see that the sentence accuracy for *ws13* in CFG mode using tagged preterminals (Table 7.5) is marginally lower than the accuracy in unrestricted mode, and the similar phenomenon is observed for *cb* in CFG mode using gold standard preterminals. Although these differences are too subtle to draw any conclusions regarding which parsing mode is the best-performing, we can explain the underlying mechanism that causes differences in the sentence accuracy between these two parsing modes: If the sentence accuracy in CFG mode is lower than in unrestricted mode, this implies that the CFG approximation blocked correct transitions predicted by the parsing oracle more frequently than it blocked an incorrect prediction and led the parser to the 'correct' transition. We find the opposite result for *cb* using tagged preterminals, and *ws13* using gold standard preterminals, suggesting that for these test configurations, the CFG approximation contributed positively to the sentence accuracy.

The transition accuracy in these parsing modes is further discussed in Section 7.4.4. The expectation in implementing the CFG approximation was that the overall quality of the parse derivations would increase, but Tables 7.4 and 7.5 at least do not demonstrate a consistent trend supporting this assumption.

We see a significant drop in parsing accuracy from Table 7.4 (gold standard preterminals) to Table 7.5 (tagged preterminals) when pairwise comparing the accuracy for the three parsing modes. Even though we have reached a high level of accuracy for the supertagger, the sentence accuracy for the supertagger was 63.7% (see Section 4.5) for *ws13* and 34% for *cb*, leaving roughly 1/3 and 2/3 of the sentences with inaccurate preterminals. Although EVALB does not take preterminals into account when evaluating F-measure and sentence accuracy, an incorrect preterminal is very likely to propagate mistake(s) to the constituent structure.

Finally, we see a substantial drop in performance between the in-domain *ws13* and the out-of-domain *cb* data sets, which is particularly noticeable for the sentence accuracy. Not only is *cb* out-of-domain, but the average sentence length for this data set is more than four words longer than *ws13*, and both of these factors are likely to contribute to this effect.

7.4.3 Semantic Evaluation

While the syntactic evaluation carried out in Section 7.4.2 gives a strong insight into the parser's capability to replicate the gold standard derivation, the semantic evaluation will assess the semantic representation based on the

Elementary Dependencies derived from the parsed analysis, as presented in Section 7.1.2.

Tables 7.6 and 7.7 present the EDM scores for the deterministic CuteForce configuration. EDM_A and EDM_P scores are presented with precision *($E_{A/P}$-P)*, recall *($E_{A/P}$-R)* and F-measure *($E_{A/P}$-F1)*, and an additional F-measure combining the scores from EDM_A and EDM_P, presented as *(E_{AP}-F1)*.[4]

Data	Mode	E_A-P	E_A-R	E_A-F1	E_P-P	E_P-R	E_P-F1	E_{AP}-F1
ws13	Unr	85.2	83.6	84.9	94.3	94.5	94.4	91.1
ws13	CFG	88.9	60.0	71.6	95.3	65.2	77.4	75.5
ws13	Unify	94.7	33.3	49.2	97.5	33.5	49.9	49.7
cb	Unr	84.2	82.1	83.1	93.9	94.7	94.3	90.8
cb	CFG	88.1	53.1	66.2	95.5	58.8	72.8	70.8
cb	Unify	94.2	22.1	35.8	98.0	23.2	37.6	37.0

Table 7.6: Semantic Evaluation of *ws13* and *cb* using gold standard preterminals.

Data	Mode	E_A-P	E_A-R	E_A-F1	E_P-P	E_P-R	E_P-F1	E_{AP}-F1
ws13	Unr	81.3	79.4	80.4	91.7	91.7	91.7	87.9
ws13	CFG	85.2	56.0	67.6	92.8	61.9	74.3	72.0
ws13	Unify	91.8	29.3	44.4	95.5	29.8	45.4	45.1
cb	Unr	78.5	76.1	77.2	90.3	90.6	90.4	86.3
cb	CFG	82.8	48.0	60.8	91.9	55.1	68.9	66.4
cb	Unify	89.6	16.6	27.9	95.1	17.6	29.7	29.1

Table 7.7: Semantic Evaluation of *ws13* and *cb* using supertagged preterminals.

Tables 7.6 and 7.7 largely confirm the trend that could be observed in the syntactic evaluation. Precision increases and recall drops in CFG and unification mode. Additionally, the observed drops in F-measure between the gold standard and tagged preterminals are very similar to the drop we see in the syntactic analysis. These observations indicate that the syntactic and semantic evaluations are in agreement, and it is especially interesting to observe that the PARSEVAL F-measures in Tables 7.4 and 7.5 are very similar to the EDM_A F-measure in the semantic evaluation, both concerning the actual values, and how they relate to one another in the alternative configurations. To exemplify, we find a PARSEVAL F-measure of 81.7% and 68.1% for the unrestricted and CFG modes using tagged preterminals (*ws13*), whereas the corresponding EDM_A F-measures in Table 7.7 are 80.4% and 67.6%.

[4]The E_{AP}-F1 metric is computed over the total count of EDM triples of type argument EDM_A and property EDM_P.

7.4.4 Error Analysis

Tables 7.8 to 7.10 present the number of correct and incorrect transitions executed by CuteForce, collected from the parsing of *ws13* using tagged preterminals for the three available parsing modes. Upon the first transition that is inconsistent with the gold standard, no further statistics are collected for the current sentence, since an incorrect transition chooses an incorrect search path, and beyond this point there is no longer a well-defined notion of correct and incorrect parsing decisions.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4725	91	94	0
UNIT	63	4709 (104)	12	0
PASSIVE	70	36	2772 (36)	0
ACCEPT	0	0	0	274 (5)

Table 7.8: Confusion matrix for *ws13* using supertagged preterminals, unrestricted parsing mode. In total 12,572 transitions.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4719	94	94	0
UNIT	63	4707 (100)	14	0
PASSIVE	70	36	2765 (36)	0
ACCEPT	0	0	0	273 (5)

Table 7.9: Confusion matrix for *ws13* using supertagged preterminals, CFG approximation mode. In total 12,562 transitions.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4749	90	99	0
UNIT	63	4734 (105)	19	0
PASSIVE	68	36	2824 (20)	0
ACCEPT	0	0	0	281 (4)

Table 7.10: Confusion matrix for *ws13* using supertagged preterminals, unification mode. In total 12,682 transitions.

In the confusion matrices 7.8 to 7.10, the gold standard transitions are represented horizontally (in **bold**), whereas the parsed transitions (*italic*) are represented vertically. Hence, for Table 7.8 we see that the DETACH transition was accurately executed 4,725 times, whereas it was incorrectly executed 185 times. In 91 of these events the correct transition was UNIT, whereas PASSIVE was the correct transition in 94 times. For the parameterized transitions we distinguish between a correctly predicted transition, and a correctly predicted transition and parse label. The numbers in parenthe-

ses for the correctly predicted parameterized transitions refer to the number of incorrectly predicted parse labels, i.e. ERG rules or start symbols. We should note that for start symbol assigned by the ACCEPT parameter is not evaluated in the syntactic and semantic analyses, and will hence not be counted as a parsing mistake. For the ACCEPT transition we also note that it is always predicted with a 100% accuracy, disregarding incorrect parameters/start symbol. We assume this is due to the fact that the ACCEPT transition is in itself heavily constrained by the transition system, and will only occur in very specific configurations, e.g. after a Head+Subject rule has been assigned to the HPSG derivation.

A successful parse of a sentence will have executed the same number of DETACH and PASSIVE transitions. In Tables 7.8 to 7.10 we see however a large bias towards DETACH transitions. This is because DETACH transitions are executed *before* the corresponding PASSIVE transition, and if an incorrect parsing decision occurs between these two transitions, only the DETACH transition will be recorded in the confusion matrix.

The total number of transitions varies for the three parsing modes since the data is collected up to the point where an incorrect search path is chosen. The highest number of transitions is collected when the parser is operating in unification mode (Table 7.10), in total there were 110 and 120 transitions more than for unrestricted and CFG mode, respectively. Compared to parsing in unrestricted mode this figure is as expected – in unification mode only incorrect parsing decisions will be blocked. By selecting a transition that is not violating a grammatical constraint, at the expense of a transition which does, the parser is effectively increasing its chances of remaining on the correct search path, in which case further correct transitions will be recorded in this table.

Perhaps more interestingly, we see that when parsing in unrestricted mode, CuteForce records 10 transitions more than in CFG approximation mode. This indicates that one or more correct parsing decisions by the oracle have been blocked by the CFG approximation filter. In itself this could imply that the CFG approximation is too restrictive. Coupled with the statistics for the syntactic evaluation in Section 7.4.2, where we saw there was no consistent trend where the sentence accuracy increased when CFG filtering was applied, this seems to suggest that in its current state, the CFG filter is not contributing to improving the parsing compared to the unrestricted parsing mode. This will be further discussed in the conclusion of the chapter, Section 7.8.

Frequent Parsing Mistakes Tables 7.11 to 7.13 list the eight most frequent parsing mistakes when parsing *ws13* using tagged preterminals. Most strikingly we observe that the list is almost identical for all three parsing modes, both with respect to the incorrectly predicted transitions and their

actual frequency. This is not unexpected. We would expect the most frequent parsing mistakes to occur in contexts where alternative, incorrect transitions frequently occur in the training data. The plausibility of an alternative transition suggests that this transition, albeit being incorrect for a given context, is not in violation with a grammatical constraint, and is hence not blocked in unification mode.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	36
DETACH	UNIT(hdn_bnp_c)	23
DETACH	PASSIVE(sp-hd_n_c)	20
PASSIVE(hd-cmp_u_c)	DETACH	19
DETACH	PASSIVE(hd-cmp_u_c)	17
UNIT(hdn_bnp_c)	DETACH	16
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	16
PASSIVE(aj-hdn_norm_c)	DETACH	15

Table 7.11: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, unrestricted parsing mode.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	36
DETACH	UNIT(hdn_bnp_c)	24
DETACH	PASSIVE(sp-hd_n_c)	20
PASSIVE(hd-cmp_u_c)	DETACH	18
UNIT(hdn_bnp_c)	DETACH	16
DETACH	PASSIVE(hd-cmp_u_c)	16
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	15
PASSIVE(aj-hdn_norm_c)	DETACH	15

Table 7.12: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, CFG approximation mode.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	36
DETACH	UNIT(hdn_bnp_c)	23
DETACH	PASSIVE(sp-hd_n_c)	21
PASSIVE(hd-cmp_u_c)	DETACH	20
DETACH	PASSIVE(hd-cmp_u_c)	20
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	16
UNIT(hdn_bnp_c)	DETACH	15
PASSIVE(aj-hdn_norm_c)	DETACH	15

Table 7.13: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, unification mode.

In Tables 7.11 to 7.13 we observe that, excluding the prediction of an incorrect comma rule, all parsing mistakes are related to nominal heads, and

can be summarized as noun phrase ambiguity mistakes.[5] These syntactic constellations are notoriously difficult for a wide range of parsing platforms. In the HPSG universe, Toutanova et al. (2002) describe experiments on parse disambiguation using the Redwoods treebank, where the error analysis indicated that "PP attachments were the single most common error type" (Toutanova et al., 2002, p. 261). Further, the quality evaluation on WikiWoods confirmed and elaborated on this tendency:

> Those items judged as nearly correct contained one or at most two minor errors which did not materially affect the overall meaning of the utterance; the errors were typically misbracketing within a complex nominal compound, misattachment of a modifying prepositional phrase, or an infelicitous coordination bracketing (Flickinger et al., 2010, p. 1670)

Considering that 80% of the training data employed by CuteForce consists of parsed WikiWoods data, it is overwhelmingly likely that these training derivations contain a relatively high proportion of parsing errors which are directly related to noun phrase ambiguities, which, added to the inherent difficulties these syntactic constructions typically pose, further reinforces this predisposition.

The most frequent parsing mistake by CuteForce, namely executing DETACH rather than PASSIVE(aj-hdn_norm_c), suggests that the parsing oracle expects an incorrect nominal attachment. We will in the following exemplify this for sentence (1), where CuteForce committed this mistake (CuteForce operated in unrestricted parsing mode using tagged preterminals). Although sentence (1) is grammatical, one can argue that it is stylistically awkward, possibly a contributing factor to why it was parsed incorrectly by CuteForce.

(1) Substantial efforts have been devoted in the last decade to the test and evaluation of speech recognition in fighter aircraft. (from *ws13*)

In Figure 7.8 we see that CuteForce fails to identify *last decade* as a nominal head with a preceding adjective (aj-hdn_norm_c), but is rather executing a DETACH transition upon having correctly predicted a UNIT(n_sg_ilr) transition. A DETACH transition in this context seems to suggest that the subsequent preposition *to* has mislead the parser to falsely expect a prepositional phrase attachment, an error which would be consistent with the frequently observed parsing mistakes reported in Flickinger et al. (2010) and Toutanova et al. (2002). In Figure 7.9 we see a gold standard subtree for sentence (1), where *last decade* has a correct analysis.

[5] See `http://moin.delph-in.net/ErgRules` for a complete overview of the ERG rule inventory.

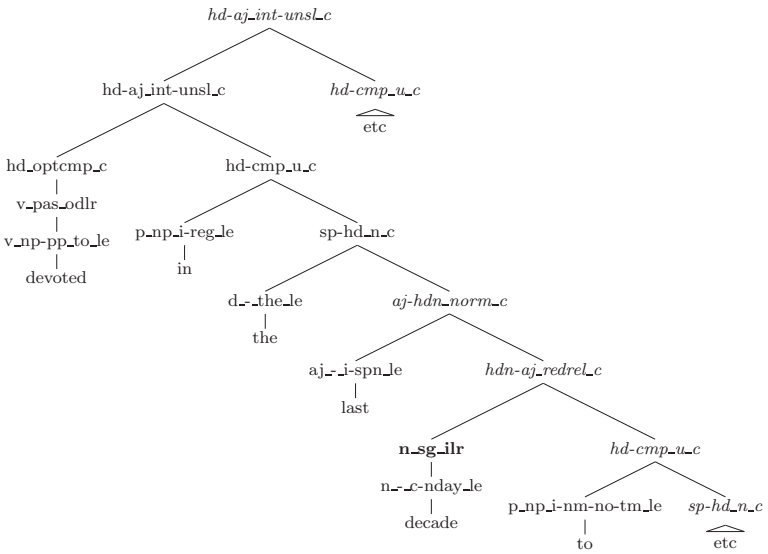

Figure 7.8: Parsed subtree from sentence (1), where an incorrect DETACH transition is executed after the transition UNIT(n_sg_ilr) **(bold)** was correctly predicted. Rule instantiations assigned after the parsing mistake are represented in *italic*.

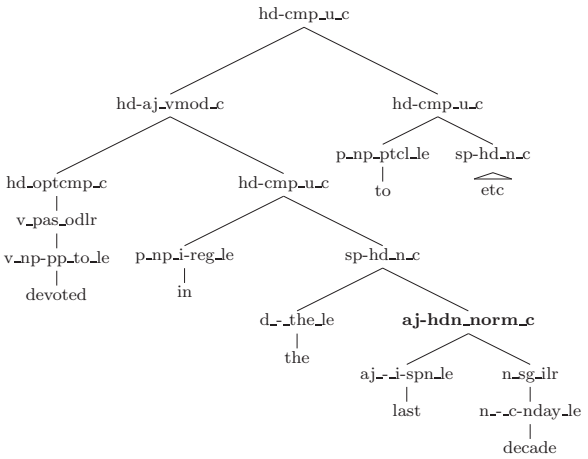

Figure 7.9: Gold standard subtree from sentence (1), where PASSIVE(aj-hdn_norm_c) **(bold)** is correctly predicted.

Based on the previous experiences with parsing and parse disambiguation on ERG, the distribution of parsing mistakes that we have observed for CuteForce seems unsurprising. However, considering that the training data is itself biased towards these parsing mistakes, we expect that this has further contributed to an increase in the number of parsing errors related to complex nominal phrases.

Over- and Undergeneration A highly skewed dataset is typically inclined to trigger overgeneration for the most frequent class(es) in the training data, implying that the majority class is predicted more frequently than it should. In our training data, the DETACH transition is clearly the most frequent transition, appearing in approximately 31% of the classification instances in the training data, being far more frequent than any other of the 274 classes which are found in the training data, keeping in mind that a class predicted by the oracle consists of the transition, and a potential parse label.

There are a wide range of methods for measuring the observed degree of under- and overgeneration of a class. In the following we will only evaluate the parsing oracles ability to predict the main transition (DETACH, UNIT, PASSIVE, and ACCEPT), and disregard the parse label for the parameterized transitions. Hence, e.g. a UNIT transitions with an incorrect parse label will in this evaluation be regarded a *True Positive*, because the main transition (UNIT) was predicted correctly.

	OG	UG
DETACH	3.8	2.7
UNIT	1.6	2.6
PASSIVE	3.6	3.6
ACCEPT	0	0

Table 7.14: The degree of overgeneration (OG) and undergeneration (UG) for the transitions in Table 7.8, i.e. the unrestricted parsing mode.

In Table 7.14 we present the degree of overgeneration as the proportion of false positives for a given class over the sum of the true positives and false negatives, i.e. the total sum of correct transitions in the data set: $\frac{FP}{TP+FN}$. Further, the degree of undergeneration is computed as $\frac{FN}{TP+FN}$, i.e. the proportion of predictions where the classifier failed to predict the gold standard class. These two scores should be seen in context. If the degree of overgeneration (OG in Table 7.14) is much higher than the corresponding degree of undergeneration (UG), this suggests that the classifier is overgenerating this class. The opposite effect would be indicative of an undergeneration of a prediction class. To exemplify, for the DETACH transition, we computed the

degree of overgeneration and undergeneration to be $100 * \frac{91+94}{63+70+4725} = 3.8$, and $100 * \frac{63+70}{63+70+4725} = 2.7$, respectively.

In Table 7.14 we provide the degree of over- and undergeneration for the transition statistics in the confusion matrix 7.8, i.e. the unrestricted parsing mode. We have chosen to exemplify with this parsing mode, since this mode is better equipped to reflect a potential bias of the oracle towards certain transitions, compared to parsing in CFG and unification modes, where a larger proportion of the search space will be pruned.

Based on the aforementioned metric, we see a slight tendency towards an overgeneration of the DETACH transition, and a similar disposition towards an undergeneration of the UNIT transition. The degrees of over- and undergeneration of the PASSIVE transition are identical in this configuration. We find these results to be very encouraging. The small differences between the degree of over- and undergeneration in Table 7.14 imply that the parsing model, i.e. the oracle, in overall appears very balanced, suggesting that the learning algorithm we have used is very robust against imbalanced training data.[6] If we had seen a larger bias towards the DETACH transition, we would have considered strategies frequently used to address challenges associated with a highly-skewed training data, possibly though the use of downsampling or cost-sensitive learning (McCarthy et al., 2005; Elkan, 2001). However, rather than pursuing such strategies, we evaluated two-stage classification. A two-stage classification setup would maintain a more balanced distribution between the three main transitions, UNIT, DETACH and PASSIVE, hence allowing us to evaluate the oracle when trained on a more balanced data set. The two-stage classification setup will be evaluated in the following section.

7.5 Two-Stage Classification

While CuteForce's default classification oracle, evaluated in the previous section, applies one single classification for each transition, effectively applying Multiclass SVM classification over all applicable transitions, MaltParser (Nivre et al., 2007) also supports a two-stage multiclass classification, where the first stage selects a transition type, and in the event of a parameterized transition, a subsequent classifier selects the transition label. This strategy was also evaluated for CuteForce, which we will report on in this section.

A two-stage classification pipeline allows for the use of more training data, because training instances will be split among multiple training models, where each classifier will have fewer prediction classes (i.e. transitions), hence being less computationally expensive to train if the amount of training

[6] As our learning algorithm, we applied LibLinear (Fan et al., 2008), using the SVM multiclass classification of Crammer and Singer's formula (Keerthi et al., 2008; Crammer and Singer, 2002; Vapnik, 1995), as discussed in Section 5.3

data is constant. In our evaluation of this method, we trained one classifier that selects between the four main transitions (DETACH, UNIT, PASSIVE and ACCEPT), and additional training models to select the transition labels for the parameterized transitions (UNIT, PASSIVE and ACCEPT). Applying this configuration, we successfully trained four models, using 500,000 sentences in our training data, i.e. an increase of 350,000 sentences compared to the amount of training data employed by the single-stage configuration. The training data was expanded by including addition sentences from WikiWoods to the pool of training derivations.

Evaluation on the development data set revealed that a two-stage classification scheme did not reach a higher accuracy than the default single-stage setup. The differences between the two setups were however not large. CuteForce is implemented with limited support for two-stage classification, specific to the deterministic unrestricted and CFG parsing mode. In the following, we present the parsing results and error analysis for the two-stage classification setup.

7.5.1 Parser Statistics

Data	Mode	COV	CON	PT	PT-P	PT-F	SL-P	SL-F	TA
ws13	Unr	100	50.8	12.0	12.0	N/A	14.1	N/A	N/A
ws13	CFG	77.1	50.4	11.1	9.8	15.9	11.8	21.8	N/A
cb	Unr	100	30.4	15.6	15.6	N/A	18.4	N/A	N/A
cb	CFG	63.4	28.6	12.8	15.3	13.7	16.3	22.0	N/A

Table 7.15: Parser Statistics for the two-stage CuteForce configuration using gold standard preterminals for the test sets *ws13* and *cb*.

Data	Mode	COV	CON	PT	PT-P	PT-F	SL-P	SL-F	TA
ws13	Unr	100	47.1	10.9	10.9	N/A	14.1	N/A	95.4
ws13	CFG	76.6	46.9	10.7	9.5	14.8	11.8	21.8	95.8
cb	Unr	100	26.9	13.9	13.9	N/A	18.4	N/A	91.6
cb	CFG	60.1	25.3	12.1	15.0	13.3	16.0	21.9	92.6

Table 7.16: Parser Statistics for the two-stage CuteForce configuration using supertagged preterminals for the test set *ws13* and *cb*.

Tables 7.15 and 7.16 present the parser statistics for *ws13* and *cb* applying the deterministic two-stage oracle configuration. Compared to the single-stage oracle configuration presented in Section 7.4.1, we see that coverage and the proportion of consistent derivations have dropped for all setups in Tables 7.15 and 7.16. This is especially apparent in the CFG mode, where we find that for *cb* (tagged preterminals) the coverage has dropped from 68% to 60.1%.

We should also note that although it seems the parsing times are unaffected, this configuration requires much more memory, since the number of features is higher, due to the fact that the size of the training data is more than three times larger than the default single-stage oracle configuration.

7.5.2 Syntactic Evaluation of Two-Stage Classification

Data	Mode	COV	PR	RE	F1	SA
ws13	Unr	100	84.4	85.0	84.7	40.3
ws13	CFG	77.1	88.6	57.3	69.6	40.6
cb	Unr	100	82.5	82.9	82.7	20.6
cb	CFG	60.1	85.6	47.9	61.4	20.2

Table 7.17: PARSEVAL scores for *ws13* and *cb* using gold standard preterminals, using two-stage deterministic CuteForce configuration.

Data	Mode	COV	PR	RE	F1	SA
ws13	Unr	100	81.3	81.4	81.4	34.7
ws13	CFG	76.6	85.0	54.5	66.4	34.7
cb	Unr	100	77.3	77.4	77.4	12.8
cb	CFG	60.1	81.7	42.8	56.2	12.8

Table 7.18: PARSEVAL scores for *ws13* and *cb* using supertagged preterminals, using two-stage deterministic CuteForce configuration.

The scores for the syntactic evaluation of CuteForce using the two-stage oracle setup are presented in Tables 7.17 and 7.18. We observe that the F-measures are slightly higher in the single-stage configuration (Tables 7.4 and 7.5), compared to the corresponding F-measures for the two-stage oracle setup. We find that the recall/precision variations are insignificant for all recall and precision values in the unrestricted parsing mode. However, in CFG mode the single-stage classification setup reaches significantly higher recall both for the in-domain and out-of-domain data sets.

7.5.3 Semantic Evaluation of Two-Stage Classification

Tables 7.19 and 7.20 present the results for the semantic evaluation for CuteForce using the two-stage oracle setup. As can be expected, the results follow much the same pattern as the syntactic results. While the F-measures in unrestricted parsing mode are generally quite similar to the corresponding scores in the single-stage oracle setup, we find larger differences in CFG mode. To exemplify, for *ws13* (tagged preterminals, CFG mode) we observe an EDM_{AP}-F1 score of 69.8% for the two-stage setup, compared to 72% in the corresponding single-stage setup.

Data	Mode	E_A-P	E_A-R	E_A-F1	E_P-P	E_P-R	E_P-F1	E_{AP}-F1
ws13	Unr	83.5	82.8	83.2	93.4	94.3	93.8	90.3
ws13	CFG	88.9	56.4	69.0	95.4	61.1	74.5	72.7
cb	Unr	82.9	81.8	82.3	93.5	94.5	94.0	90.3
cb	CFG	87.2	47.2	61.2	95.1	53.0	68.1	65.9

Table 7.19: Semantic Evaluation of *ws13* and *cb* using gold standard preterminals using two-stage deterministic CuteForce configuration.

Data	Mode	E_A-P	E_A-R	E_A-F1	E_P-P	E_P-R	E_P-F1	E_{AP}-F1
ws13	Unr	79.9	78.7	79.3	91.2	91.3	91.3	87.3
ws13	CFG	84.5	53.2	65.3	92.6	59.0	72.1	69.8
cb	Unr	77.6	76.1	76.8	90.1	90.5	90.3	86.1
cb	CFG	82.4	41.5	55.2	91.8	47.7	62.8	60.4

Table 7.20: Semantic Evaluation of *ws13* and *cb* using supertagged preterminals using two-stage deterministic CuteForce configuration.

7.5.4 Error Analysis

From the error analysis of the default deterministic configuration of CuteForce (Section 7.4.4), we remember that we saw a slight overgeneration of the DETACH transition compared to the other transitions. A plausible explanation for this behavior is that the training data is skewed towards this transition because the DETACH transition is by far the most frequent outcome class among the instances in the training data.

Since the two-stage classification setup will initially select among the four main transition categories, without simultaneously predicting the parse label for the parameterized transitions, we would not experience the same bias towards DETACH transitions in the training data. In a two-stage classification setup the number of DETACH and PASSIVE transitions in the training data is identical. Specifically, in the pool of training data used in this setup, we find 31.1% DETACH and PASSIVE transitions, 35.2% UNIT transitions and 2.5% ACCEPT transitions.

The confusion matrices 7.21 and 7.22 provide the transition statistics for CuteForce parsing *ws13* using tagged preterminals and applying a deterministic two-stage classification setup. We observe that the precision of predicting the parse labels, i.e. the parameters for the UNIT, PASSIVE and ACCEPT transitions, is slightly higher for the two-stage classification than the corresponding figures for the single-stage setup. The precision scores for the prediction of the UNIT and PASSIVE parse labels in Table 7.21 are $\frac{4587-88}{4587} = 0.981$ and $\frac{2714-26}{2714} = 0.991$, respectively. Although these figures appear higher than the corresponding scores for the single-stage classification setup (0.971 and 0.987, see Table 7.8), the scores are not directly comparable, because they are not evaluated over the same set of transition

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4599	103	90	0
UNIT	63	4587 (88)	13	0
PASSIVE	88	42	2714 (26)	0
ACCEPT	0	0	0	271 (1)

Table 7.21: Confusion matrix for *ws13* using supertagged preterminals, unrestricted parsing mode, two-stage classification. In total 12,570 transitions.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4598	103	91	0
UNIT	63	4584 (90)	14	0
PASSIVE	85	39	2719 (25)	0
ACCEPT	0	0	0	271 (1)

Table 7.22: Confusion matrix for *ws13* using supertagged preterminals, CFG approximation mode, two-stage classification. In total 12,567 transitions.

predictions.

Frequent Parsing Mistakes Tables 7.23 and 7.24 list the eight most frequent parsing mistakes for the two-stage classification setup when parsing *ws13* using tagged preterminals. Comparing the findings with the corresponding list for the single-stage oracle setup (Tables 7.11 and 7.12) we see that the overall trend suggests that there are no larger dissimilarities between the most common parsing mistakes, although the frequencies of the mistakes are slightly different.

Although the training models and the classification setup differs, the most frequent parsing mistake is identical for both classification setups, and we see that the same parsing mistakes reoccur for the various setups and classification configurations for CuteForce, although their frequencies vary. We particularly note two frequent occurrences of a DETACH undergeneration, where PASSIVE(hd-cmp_u_c) and PASSIVE(aj-hdn_norm_c) are executed, rather than the correct DETACH transition. This tendency could be triggered because the classifier predicting the main transition were less biased towards DETACH transitions in this setup, and this will be discussed in the following paragraph. However, we cannot rule out the possibility that the classifier which selects the PASSIVE parse label has become more biased towards these parse labels in the event where an incorrect PASSIVE transition was predicted by the initial classifier.

Over- and Undergeneration From Table 7.14 we remember that the parsing model for the single-stage classification configuration appeared quite

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	35
PASSIVE(hd-cmp_u_c)	DETACH	25
DETACH	UNIT(hdn_bnp_c)	24
PASSIVE(aj-hdn_norm_c)	DETACH	21
DETACH	PASSIVE(sp-hd_n_c)	16
UNIT(hdn_bnp_c)	DETACH	15
DETACH	UNIT(hd_optcmp_c)	15
DETACH	UNIT(hdn_bnp-pn_c)	13

Table 7.23: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, unrestricted parsing mode, two-stage classification.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	35
DETACH	UNIT(hdn_bnp_c)	25
PASSIVE(hd-cmp_u_c)	DETACH	24
PASSIVE(aj-hdn_norm_c)	DETACH	21
DETACH	PASSIVE(sp-hd_n_c)	16
UNIT(hdn_bnp_c)	DETACH	15
DETACH	UNIT(hd_optcmp_c)	15
DETACH	UNIT(hdn_bnp-pn_c)	14

Table 7.24: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, CFG approximation mode, two-stage classification.

	OG	UG
DETACH	4.1	3.2
UNIT	1.6	3.1
PASSIVE	4.6	3.7
ACCEPT	0	0

Table 7.25: The degree of over- and undergeneration for the prediction of the transitions in Table 7.21, i.e. the unrestricted parsing mode, using two-stage classification.

balanced, but we saw a slight overgeneration of the DETACH transition and an undergeneration of the UNIT transition. A plausible expectation for a two-stage classification setup was that the more balanced training data would lead to less over/undergeneration than we saw for the single-stage classification setup in Section 7.4. In Table 7.25 we do however not find any evidence to support this hypothesis. The degree of over- and undergeneration of the DETACH transition in the single-stage setup were 3.8% and 2.7%, respectively. The corresponding scores in the two-stage setup are

4.1% and 3.2%, and hence very similar to the relation we saw for the single-stage setup. Further we see that the UNIT transition undergenerates more frequently and the PASSIVE transition overgenerates much more than we found when using the single-stage parsing model.

Given that this statistic is limited to *ws13*, we evaluated the transition accuracies when parsing on *cb*, keeping the remaining CuteForce-specific configurations (e.g. parsing mode) identical. The transition statistics for this data set, presented in Appendix B, supports our conclusion, suggesting that the imbalanced training data used in the training of the single-stage parsing oracle is not more prone to overgenerate the majority class, compared to the two-stage classification setup. On the contrary, we find that the degree of over- and undergeneration of the DETACH transition is 4.7% and 3.0% for the two-stage classification setup parsing *cb* (presented in Table B.4, Appendix B), while the corresponding over- and undergeneration scores for the single-stage setup are 4.5% and 3.0% (Table B.2).

7.5.5 Summary and Conclusion for Two-Stage Classification

The parsing results for the two-stage classification suggested that this oracle configuration was obtaining similar to lower parser accuracies compared to the single-stage classification setup. Perhaps more interestingly, although the two-stage classification setup trains on much more balanced data, we did not find any evidence to suggest that a single-stage classifier is more likely to overgenerate the majority class, compared to the two-stage classifier, that is trained on much more balanced data.

The difference in parsing performance between these two configurations was small, both with respect to parser efficiency, coverage, and parsing accuracy. While we were not able to detect significant differences in the recall/precision scores in unrestricted parsing mode, we found the recall using the single-stage oracle setup to be significantly higher in CFG mode, compared to using the two-stage classification setup. A two-stage classification model facilitated the use of more training data. Trained on the same amount of data as the single-stage parsing oracle, the two-stage classifier performed much worse. We saw learning curves quite similar to the single-stage classification configuration, where the parsing accuracy of CuteForce operating in the two-stage setup increased gradually up until we reached the maximum number of dimensions that LibLinear facilitated, at approximately 500,000 training sentences.

Considering that the most competitive setup of CuteForce employing a two-stage classification setup is more computationally complex and expensive than the single-stage configuration, requiring more training data, memory and multiple training models, and without offering increased accuracy, we decided to abandon this branch, and did therefore not include support for unification and near-deterministic parsing for the two-stage classification

setup.

7.6 Near-Deterministic Parsing

In this section, we will present parsing results for CuteForce operating in near-deterministic mode using the Optimistic Backtracking overlay presented in Chapter 6. Both the ranking-based backtracking approach (presented in Section 6.5), and the alternative baseline approach (see Section 6.6) will be subject for a syntactic and semantic evaluation.

7.6.1 Parsing Statistics

Tables 7.26 and 7.27 present the parsing statistics for CuteForce in near-deterministic mode. From the parsing statistics for the deterministic configuration (Section 7.4.1) the table is expanded with one column. CuteForce supports two backtracking modes *(BM)*, namely *Ranking-based (R)* and *Baseline (B)*. The rest of the table follows the same conventions as for the deterministic configuration. As opposed to the evaluation of the deterministic configuration, CuteForce is only evaluated in CFG and unification mode, i.e. the two parsing modes that may experience a parsing failure.

Data	BM	Mode	COV	CON	PT	PT-P	PT-F	SL-P	SL-F	TA
ws13	R	CFG	94.0	54.4	27.1	21.7	111.4	13.3	26.3	N/A
ws13	B	CFG	91.5	54.1	20.5	17.5	56.6	13.1	24.8	N/A
ws13	R	Unify	57.5	57.5	2.0k	1.4k	2.7k	8.6	21.5	N/A
ws13	B	Unify	57.2	57.2	2.8k	2.4k	3.5k	8.6	21.5	N/A
cb	R	CFG	84.4	32.5	36.7	28.5	81.0	17.3	24.1	N/A
cb	B	CFG	85.2	32.7	25.7	23.5	38.3	17.3	24.7	N/A
cb	R	Unify	37.1	37.1	3.0k	2.3k	3.5k	12.3	22.0	N/A
cb	B	Unify	36.4	36.4	3.4k	2.8k	3.8k	12.3	21.8	N/A

Table 7.26: Parser Statistics for the near-deterministic CuteForce configuration using gold standard preterminals for the test set *ws13* and *cb*.

Data	BM	Mode	COV	CON	PT	PT-P	PT-F	SL-P	SL-F	TA
ws13	R	CFG	92.4	51.5	24.4	17.5	107.7	13.0	27.2	95.7
ws13	B	CFG	90.5	50.8	18.6	16.0	43.7	13.0	24.8	95.6
ws13	R	Unify	53.3	53.3	2.3k	1.7k	2.9k	8.2	20.8	96.8
ws13	B	Unify	53.1	53.1	3.1k	2.5k	3.6k	8.3	20.7	96.8
cb	R	CFG	84.5	26.4	35.2	27.6	76.7	17.4	23.8	91.9
cb	B	CFG	82.2	26.9	23.9	20.5	39.7	17.0	24.5	91.9
cb	R	Unify	30.4	30.4	3.3k	2.5k	3.6k	11.9	21.2	94.7
cb	B	Unify	29.9	29.9	3.5k	2.9k	3.8k	11.8	21.2	94.6

Table 7.27: Parser Statistics for the near-deterministic CuteForce configuration using supertagged preterminals for the test set *ws13* and *cb*.

Compared to the parser statistics for the deterministic configuration (see Tables 7.2 and 7.3), we see that coverage and the proportion of consistent sentences have increased for all configurations. This is to be expected, because backtracking will only be launched in a situation where the sentence would have been abandoned due to a parse failure in deterministic mode. Hence, rather than conceding a parse failure for the sentence, the backtracker will try to repair the parsing mistake that was (presumably) committed up until the point of parsing failure. Therefore, we know that the coverage and consistency scores will be *at least* as good as the scores for the deterministic configuration, given that the alternative to backtracking is that parsing of the sentence fails.

With respect to the two applicable parsing modes, CFG and unification mode, we see that coverage and consistency increased far more in CFG mode, with an increase in coverage from approximately 80% to 90% for *ws13*. In unification mode, this increase is however negligible. For *ws13* we see that the coverage goes from 56.7% to 57.5% (gold standard preterminals) and 52.6% to 53.3% (tagged preterminals) when applying the ranking-based backtracking method. Further, we see that parsing times increase in the range of 100-150% in CFG mode compared to the deterministic configuration. We see more variation in the parsing times in unification mode, and observe that the parsing time increase in some configurations is marginal or non-existent (e.g. for *ws13*, using gold standard preterminals), and up to 75% in other configurations.

For the two alternative backtracking methods, ranking-based and baseline, we find the increase in coverage to be larger for the ranking-based approach for all configurations we have evaluated (Tables 7.26 and 7.27), except for *cb* in CFG approximation mode using gold standard supertags. Overall, we see that the differences between the two backtracking approaches are small. The most pronounced difference is found for the in-domain *ws13* data in CFG mode. Considering that the backtracker is trained on Wikipedia data, it should not be surprising that the backtracker exhibits higher performance on this data. Although the general trend is that the ranking-based method increased the coverage more than the baseline method, we find the results from Tables 7.26 and 7.27 to be inconclusive, considering that the differences between the two approaches are rather small.

7.6.2 Syntactic Evaluation

For the syntactic evaluation of CuteForce applying the near-deterministic overlay, we use the same metrics as for the deterministic configuration in Section 7.4.2. The parsing scores for the two alternative backtracking approaches, ranking-based and baseline backtracking are presented in Table 7.28 (gold standard preterminal) and Table 7.29 (tagged preterminals). *BM* in the two tables refers to the two backtracking methods evaluated:

(R)anking-based and (B)aseline.

Data	BM	Mode	COV	PR	RE	F1	SA
ws13	R	CFG	94.0	86.1	76.7	81.1	42.9
ws13	B	CFG	91.5	87.4	74.4	80.4	43.4
ws13	R	Unify	57.5	94.3	33.9	49.9	43.8
ws13	B	Unify	57.2	94.2	33.8	49.8	43.8
cb	R	CFG	84.4	83.7	67.0	74.5	21.8
cb	B	CFG	85.2	84.3	67.8	75.1	21.6
cb	R	Unify	37.1	91.8	22.8	36.5	22.7
cb	B	Unify	36.4	91.6	22.4	36.0	22.7

Table 7.28: Results for *ws13* and *cb* using Optimistic Backtracking and gold standard preterminals.

Data	BM	Mode	COV	PR	RE	F1	SA
ws13	R	CFG	92.4	82.8	70.7	76.3	36.2
ws13	B	CFG	90.5	83.9	69.9	76.3	36.6
ws13	R	Unify	53.3	90.9	29.3	44.3	36.1
ws13	B	Unify	53.1	91.1	29.5	44.6	36.1
cb	R	CFG	84.5	78.6	63.2	70.0	13.7
cb	B	CFG	82.2	79.3	60.8	68.9	13.7
cb	R	Unify	30.4	88.0	17.3	28.9	12.5
cb	B	Unify	29.9	87.6	17.3	28.9	12.8

Table 7.29: Results for *ws13* and *cb* using Optimistic Backtracking and supertagged preterminals.

Compared to the corresponding results for the deterministic configuration (see Tables 7.4 and 7.5), we see that the substantial increase in coverage for the CFG mode contributes to a higher F-measure for all applicable configurations – this follows naturally since the reduction of parsing failures directly leads to higher recall. The improved recall will in turn lead to a lower precision, since a number of presumably 'hard' sentences which in the deterministic configuration were skipped, through backtracking are successfully parsed. This subset of sentences which are now parsed has a lower precision score than the precision for the subset of sentences which were parsed in the deterministic configuration, hence lowering the average precision for the near-deterministic configurations. For *ws13* (tagged preterminals) the F-measure increases from 68.1 to 76.3% in CFG mode. In unification mode the increases in the F-measure in the near-deterministic mode were negligible compared to the deterministic configuration.

Comparing the sentence accuracies (SA) in Tables 7.28 and 7.29 with the corresponding deterministic results, we see very small differences, with

an increase from 0 and up to 1.5 percent point. In unification mode we also notice that the sentence accuracy is identical in the deterministic and near-deterministic configuration in all but one configuration (for *cb*, tagged preterminals).

The lack of a conclusive positive effect for CuteForce in unification mode when backtracking is applied, suggests that the lightweight backtracking approach within the 'optimistic' constraints (see Section 6.4) is not sufficient to substantially increase the coverage in this parsing mode. There are at least two intuitions for this: First, the grammatical constraints of a unification-based grammar do not leave much leverage when the parser goes 'off track', and chooses a search path which is inconsistent with the gold standard. The very high precision in unification mode attests to this, suggesting that the tolerance for imprecise parsing decisions is low. Second, the backtracker's accuracy in identifying the incorrect transition, and replacing it with the gold standard transition is substantially lower in unification mode than in CFG mode (this was discussed in Section 6.7). From Table 6.3 we saw that the success rate for an ideal backtrack was less than 10% when applying ranking-based backtracking.[7] Coupled with the optimistic constraints, i.e. assuming that the previous backtrack was correct, and the notion that the selection of an incorrect search path is very likely to lead to a parsing failure, it would seem that the 'optimistic' constraints are too strict in unification mode. A feasible strategy from a robustness perspective would be to relax the unification constraint if the parser reaches a parsing failure, hence continue parsing from this state in unrestricted parsing mode, ensuring a 100% coverage in a deterministic configuration. Inconsistencies in the parse derivation could upon termination be resolved with default unification. This strategy would resemble the strategy of Ninomiya et al. (2011), where default unification is applied to overwrite inconsistent constraints in the grammar. For the scope of this thesis, we have however not carried out these experiments, but have opted for using unrestricted parsing mode as the configuration that ensures 100% coverage. This design choice is further discussed in Section 8.4.5.

Between the two alternative backtracking strategies we observe only minor differences in performance. In unification mode, we find that all precision/recall differences between the two backtracking approaches are insignificant, except for the precision of *cb* using tagged preterminals. We find somewhat larger differences in CFG mode, and the significance testing indicates that the baseline backtracking approach reaches a significantly higher precision for *ws13* both for tagged and gold standard preterminals. Further, we find the precision/recall differences for *cb* using tagged preterminals to be significant, i.e. the ranking-based approach reaches a significantly

[7] By *ideal* we mean that the backtracker successfully identifies the incorrect transitions, and replaces this with the correct transition.

higher recall, and vice versa for the precision score.

For the sentence accuracies, the two backtracking methods reached the same sentence accuracy in five of the eight different configurations, and there were only subtle differences in the remaining three configurations.

7.6.3 Semantic Evaluation

Similar to the syntactic evaluation in the last section, the semantic evaluation of the near-deterministic configurations will present the parsing results for the ranking-based approach in addition to the baseline backtracking method. Table 7.30 presents the results for the near-deterministic CuteForce configurations when parsing with gold standard preterminals, whereas Table 7.31 presents the corresponding results for parsing with tagged preterminals.

Data	Mode	BM	E_A-P	E_A-R	E_A-F	E_P-P	E_P-R	E_P-F	E_{AP}-F
ws13	CFG	R	88.7	76.8	81.0	94.6	84.5	89.3	86.5
ws13	CFG	B	87.7	73.1	79.8	95.2	80.7	87.3	84.8
ws13	Unify	R	94.6	33.9	49.0	97.4	34.4	50.9	50.6
ws13	Unify	B	94.7	33.9	49.9	97.5	34.2	50.6	50.6
cb	CFG	R	85.9	67.1	75.4	94.7	75.7	84.1	81.4
cb	CFG	B	86.8	68.3	76.5	94.9	76.6	84.8	82.2
cb	Unify	R	94.1	23.0	36.9	98.0	24.2	38.8	38.2
cb	Unify	B	93.9	22.6	36.4	97.9	23.8	38.3	37.7

Table 7.30: Semantic Evaluation of *ws13* and *cb* using Optimistic Backtracking and gold standard preterminals.

Data	Mode	BM	E_A-P	E_A-R	E_A-F	E_P-P	E_P-R	E_P-F	E_{AP}-F
ws13	CFG	R	83.2	69.9	76.0	92.4	78.9	85.1	82.1
ws13	CFG	B	84.0	68.6	75.5	92.6	77.3	84.3	81.4
ws13	Unify	R	91.4	29.6	44.8	95.2	30.2	45.9	45.5
ws13	Unify	B	91.7	29.8	45.0	95.4	30.4	46.1	45.7
cb	CFG	R	80.2	62.6	70.3	91.1	73.1	81.1	77.7
cb	CFG	B	81.0	60.4	69.2	91.1	70.1	79.2	76.1
cb	Unify	R	89.8	17.4	29.2	95.3	18.6	31.1	30.5
cb	Unify	B	89.2	17.0	28.5	94.9	18.1	30.3	29.8

Table 7.31: Semantic Evaluation of *ws13* and *cb* using Optimistic Backtracking and supertagged preterminals.

With the increased coverage, the F-measures for both the EDM_A and EDM_P metrics are substantially higher in CFG approximation mode when near-deterministic backtracking is applied, compared to the corresponding results for the deterministic configuration (Section 7.4.3, Tables 7.6 and 7.7).

This increase comes as a direct result of the increased recall due to higher coverage, as we discussed in Section 7.6.2.

For the overall E_{AP}-F metric, we find that the ranking-based approach reaches a higher score than the baseline approach in three out of four configurations in Table 7.31 (tagged preterminals). For gold standard preterminals (Table 7.30) the ranking-based approach reaches the highest accuracy in two out of the four configurations. These figures are consistent with our findings for the syntactic evaluation of the two backtracking approaches, suggesting that there are only minor differences between the two backtracking strategies.

7.6.4 Error Analysis

Parallel to the evaluation of the deterministic configuration, we will in this section present an error analysis for the near-deterministic configurations of CuteForce. For evaluation, we present statistics for the parsing of *ws13* using tagged preterminals. Tables 7.32 and 7.33 hold the statistics for the ranking-based backtracking approach, and the corresponding statistics for the baseline approach are found in Tables 7.34 and 7.35.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4517	95	89	0
UNIT	60	4546 (104)	12	0
PASSIVE	74	36	2668 (31)	0
ACCEPT	0	0	0	279 (5)

Table 7.32: Confusion matrix for *ws13* using supertagged preterminals, CFG approximation mode, ranking-based backtracking. In total 12,376 transitions.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4177	101	106	0
UNIT	77	4257 (101)	23	0
PASSIVE	51	24	2398 (11)	0
ACCEPT	0	0	0	281 (2)

Table 7.33: Confusion matrix for *ws13* using supertagged preterminals, unification mode, ranking-based backtracking. In total 11,495 transitions.

Compared to the corresponding confusion matrices for the deterministic configuration (Tables 7.8-7.10), we see that the number of transitions are generally lower in the near-deterministic configuration. This is a measure of the number of transitions that have been executed from a gold standard search path. The error analysis is applied to the final sequence of transitions

which lead up to a successful parse, or a near-deterministic parsing failure. Hence, transitions that are executed in the configurations $c_n, c_{n+1}, c_{n+2}, \ldots$ will be disregarded if the backtracker later reverts to a state c_m where $m < n$, i.e. it backtracks to a configuration prior to c_n, and selects an alternative search path.

Having this in mind, it is not too surprising that the near-deterministic configuration produces fewer correct transitions than the corresponding number for the deterministic configuration. For the deterministic configuration parsing in unification mode, we find 12,682 transitions (Table 7.10). For ranking-based backtracking, the corresponding number is 11,495, indicating that the deterministic configuration has executed almost 800 more correct transitions than the near-deterministic ranking-based approach. This discrepancy, seemingly in favor of the deterministic approach, is directly related to the success rate of the backtracker. Keeping in mind that the backtracker's accuracy is less than 10% when operating in unification mode, it follows that in more than 90% of the events, the incorrect transition is not identified and replaced with the gold standard transition for a given state c. If an incorrect transition was executed in state c_f, we know that all transitions leading up to c_f are correct. However, if the backtracker incorrectly predicts the parsing mistake to be $c_n.t_n$ and $n < f$, it will effectively remove $n - f$ correct transitions from the sequence of parser transitions, as the parser will deviate from the gold standard search path in state c_n rather than c_f. A lower number of gold standard transitions in near-deterministic mode does not however affect the quality of the parsing results, considering that the reduction of correct transitions will only occur for sentences which failed in deterministic mode.

In the confusion matrices for the baseline backtracking approach, Tables 7.34 and 7.35, we find the total number of transitions to be higher than the deterministic configuration in CFG mode, and lower in unification mode. In CFG mode, the total number of transitions is 13,032 in the baseline backtracking approach – distinctly higher than for the ranking-based approach and the deterministic parsing mode. This number is not warranted based on the accuracy of the baseline backtracker, being approximately 15% (see Table 6.4), hence substantially lower than for the ranking-based approach, and we have to look beyond the statistics in the confusion matrices to uncover the reason for this behavior.

As we discussed in Section 6.6, the baseline backtracker will backtrack to the transition with the highest probability proportion of the second-highest scoring probability $P(c_c.t2)$ over the sum of the probabilities $P(c_c.t2)$ and $P(c_c.t_c)$: $\frac{P(c_c.t2)}{P(c_c.t2)+P(c_c.t_c)}$ Plausible candidates will be configurations where two alternative parsing decisions had a very similar probability. However, this will also favor configurations where the parsing oracle predicted a very flat distribution of probabilities among the applicable transitions. It is rea-

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4763	94	91	0
UNIT	63	4772 (101)	12	0
PASSIVE	68	34	2854 (35)	0
ACCEPT	0	0	0	281 (6)

Table 7.34: Confusion matrix for *ws13* using supertagged preterminals, CFG approximation mode, baseline backtracking. In total 13,032 transitions.

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	4526	95	130	0
UNIT	60	4514 (106)	14	0
PASSIVE	55	29	2587 (13)	0
ACCEPT	0	0	0	281 (2)

Table 7.35: Confusion matrix for *ws13* using supertagged preterminals, unification mode, baseline backtracking. In total 12,291 transitions.

sonable to assume that these scenarios would frequently arise in parsing configurations that are well off the gold standard search path, in which case there is no well-defined notion of a correct and incorrect transition. Hence, if an incorrect transition in configuration c_e leads to a parsing failure later on, and the backtracker reverts to a configuration c_m where $m > e$, it will have no chance of deriving the gold standard analysis, but it will follow the gold standard search path as long as the deterministic mode.

When studying the backtracking behavior for the baseline backtracker operating in CFG mode, parsing *ws13* (Table 7.34), we found that the backtracker reverted to a configuration c_m that occurred *after* c_e approximately nine times as often as it reverted to a configuration that lay prior to c_e. As we discussed earlier, it is only when the backtracker reverts to a transition that lies prior to the parsing mistake in c_e that we will experience that the number of gold standard transitions is reduced. Further, we observed that the backtracker identified and repaired an incorrect parse more frequently than it reverted to a configuration that lay prior to c_e. This explains why the near-deterministic backtracker using the baseline approach counts a higher number of correct transitions than the deterministic configuration.

Frequent Parsing Mistakes There are some differences in the distribution of the most frequent parsing mistakes in near-deterministic mode compared the the corresponding list in the deterministic configuration. We find that CuteForce operating in CFG mode applying ranking-based backtracking (Table 7.36) has a higher frequency for the most common parsing mistake, namely predicting DETACH, rather than PASSIVE(aj-hdn_norm_c),

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	41
DETACH	UNIT(hdn_bnp_c)	23
DETACH	PASSIVE(sp-hd_n_c)	20
PASSIVE(aj-hdn_norm_c)	DETACH	20
PASSIVE(hd-cmp_u_c)	DETACH	17
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	13
DETACH	PASSIVE(hd-cmp_u_c)	13
UNIT(hdn_bnp_c)	DETACH	14

Table 7.36: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, CFG approximation mode, ranking-based backtracking.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	38
DETACH	PASSIVE(hd-cmp_u_c)	22
DETACH	UNIT(hdn_bnp_c)	19
DETACH	PASSIVE(sp-hd_n_c)	19
UNIT(hdn_bnp_c)	DETACH	17
DETACH	UNIT(hdn_bnp-pn_c)	15
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	12
UNIT(hdn_bnp-pn_c)	DETACH	13

Table 7.37: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, unification mode, ranking-based backtracking.

compared to the deterministic configuration, suggesting that the backtracker in several instances has replaced correct transitions with this specific incorrect parser action. In itself this behavior is to be expected, considering that the parsing oracle in all likelihood has predicted DETACH to be the second-highest ranked transition in multiple events where PASSIVE(aj-hdn_norm_c) should have been predicted, potentially increasing the frequency for this specific parsing mistake when the backtracker falsely reverts to one of these transitions.

For the baseline backtracking method (see Tables 7.38 and 7.39), we find an even larger bias towards incorrectly selecting DETACH transitions compared to both the deterministic configuration (Tables 7.12 and 7.13), and the ranking-based backtracking approach (Tables 7.36 and 7.37). This is an indication that the parsing oracle in multiple events assigns DETACH transitions a probability which lies close to the transition with the highest probability, hence inflating this transition's chance of being a candidate to which the baseline backtracker reverts. Considering that the parsing oracle is already slightly overgenerating DETACH transitions, this behavior is quite foreseeable.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	45
DETACH	PASSIVE(hd-cmp_u_c)	36
DETACH	PASSIVE(sp-hd_n_c)	23
DETACH	UNIT(hdn_bnp_c)	21
PASSIVE(hd-cmp_u_c)	DETACH	17
UNIT(hdn_bnp_c)	DETACH	14
PASSIVE(aj-hdn_norm_c)	DETACH	13
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	12

Table 7.38: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, CFG approximation mode, baseline backtracking.

Parsed	Gold	Frequency
DETACH	PASSIVE(aj-hdn_norm_c)	46
DETACH	PASSIVE(hd-cmp_u_c)	29
DETACH	UNIT(hdn_bnp_c)	24
DETACH	PASSIVE(sp-hd_n_c)	21
UNIT(hdn_bnp-pn_c)	DETACH	14
UNIT(w_comma-nf_plr)	UNIT(w_comma_plr)	13
PASSIVE(aj-hdn_norm_c)	DETACH	13
PASSIVE(hd-cmp_u_c)	DETACH	13

Table 7.39: The eight most frequent parsing mistakes for *ws13* using supertagged preterminals, unification mode, baseline backtracking.

7.6.5 Summary and Conclusion for Near-Deterministic Parsing

In Section 7.6 we presented the parsing results of a near-deterministic configuration of CuteForce, applying the Optimistic Backtracking overlay presented in Chapter 6. The 'optimistic' constraint facilitates a backtracking framework which requires very little overhead in terms of parser complexity, with a worst-case quadratic time-complexity. We have presented parsing results for the two alternative methods proposed in Chapter 6, namely *ranking-based* backtracking, and a baseline approach which utilizes the transition pseudo-probabilities predicted by the parsing oracle.

In Section 6.7 we found that in isolation, the ranking-based approach seemed to be more accurate in predicting the correct backtracking point, compared to the baseline method. In Tables 6.3 and 6.4 we presented the accuracy, i.e. the backtrackers ability to identify and repair a incorrect parsing decision, of the individual backtracking strategies. The accuracy of the ranking-based approach was in the range of 8-20% for the various parsing modes in our evaluation in Section 7.6, the corresponding numbers for the baseline approach were 6-17%.

It is natural to evaluate a backtracking approach for a transition-based

parser by two metrics: First, the primary goal of a backtracker is to identify and repair the incorrect transition $c_e.t_e$ in the sequence of transitions up until the point of parsing failure (c_f). By repairing an incorrect parsing decision, the parser is closer to obtaining the analysis which is accordance with the gold standard. Second, although it does not successfully manage to identify and repair the incorrect parser decision, it should contribute to increased coverage by reverting the parser to a search path which will not be invalidated by the parsing mode and grammatical constraints to which the parser abides.

Coverage

For the test data evaluated in this chapter, the near-deterministic configurations (Tables 7.26 and 7.27) all reached a higher coverage than the corresponding deterministic configurations evaluated in Section 7.4.1. While we see a pronounced increase for the CFG mode, there is however only a marginal increase in near-deterministic unification mode compared to the deterministic configuration. The reason for this is twofold: First, the constraints of the unification parsing mode are much more restricted than the CFG mode, hence much less 'fault-tolerant' if the parser chooses an incorrect search path. Second, the backtracker's accuracy is substantially lower in unification mode than CFG mode, as we discussed in Section 7.6.5.

Unification Mode Based on the results presented in Tables 7.26 and 7.27, we see that Optimistic Backtracking contributes only marginally to the coverage, compared to the results for the deterministic configuration (Tables 7.2 and 7.3). The ranking-based approach yields a higher coverage than the baseline approach for all evaluated configurations in the test data, but these differences are only minor.

Based on these findings, it seems that neither of the backtracking approaches evaluated here are able to substantially improve the coverage of CuteForce when parsing in unification mode. As discussed in Section 6.3.1, the search-space for the backtracker is large, and none of the backtracking approaches we have evaluated seem to reach a sufficiently high accuracy to efficiently operate within the 'optimistic' constraints defined in Section 6.4.

To improve the coverage in unification-based parsing mode, it would seem that the optimistic constraints must be relaxed. This could be achieved by maintaining a priority queue of candidate incorrect parser decisions, hence reverting to an alternative search path if pursuing the first candidate in the priority queue did not succeed. Such a strategy would however have negative consequences for the overall complexity of the parser, and we have not pursued a further evaluation of near-deterministic parsing beyond the 'optimistic' constraints.

CFG Approximation Mode The results in Tables 7.26 and 7.27 present a substantial increase in coverage for the two backtracking approaches, compared to the results for the deterministic configuration. The increase is more pronounced for the in-domain *ws13* data set, with a parsing failure reduction of 63.1% and 53.9% for the ranking-based and baseline methods, respectively.

Based on the findings in Tables 7.26 and 7.27 both backtracking methods evaluated are successfully contributing to a substantial reduction in parsing failure.

Parsing Quality

The parsing quality has been evaluated through syntactic and semantic overlap between the parse derivation and the gold standard. We will in this evaluation disregard recall and precision, and discuss F-measure and sentence accuracy. Considering that the alternative to backtracking is parsing failure, the sentence accuracy will never yield a lower score in near-deterministic mode, compared to the corresponding deterministic configuration. We did not find it meaningful to apply pairwise significance testing to compare the parsing results between corresponding deterministic and near-deterministic configurations when we saw only marginal increases in F-measure. The way we see it, it is hard to advocate that a marginally higher recall/F-measure should justify the increased overhead that backtracking entails, even if it would happen to be 'significant' in a statistical sense.

Unification Mode For unification mode, we saw (at best) only marginal increases when backtracking was enabled. Comparing the F-measure and sentence accuracy for *ws13* using tagged preterminals (Table 7.29) with the corresponding figures for the deterministic configuration (Table 7.5), we find that although the F-measure increases slightly, the sentence accuracy remains at the same level for both backtracking methods. Considering that the accuracy of the backtracker proved to be less than 10% (see Tables 6.3 and 6.4) for the first backtracking attempt, this does not come as a surprise. The differences between the two backtracking methods are marginal. More importantly, the constraints for optimistic backtracking seem too strict in this parsing mode, and it does not seem realistic that a backtracking method will reach a high enough accuracy to efficiently contribute to a substantial increase in parsing quality compared to a deterministic configuration.

CFG Approximation Mode Due to the substantial increase in coverage, the F-measure for the syntactic and semantic evaluations for CuteForce in CFG mode increases when backtracking is applied. For the *ws13* data set using tagged preterminals, the F-measure for the syntactic evaluation

increases from approximately 68% to 76%. However, for the sentence accuracy we see only a marginal increase from 35.4% in deterministic mode to 36.2%/36.6% for the two backtracking methods. This relatively low increase must be seen in relation with the backtracking accuracy, which, although being higher than for the unification mode, does not reach a higher accuracy than approximately 15-20% in Tables 6.3 and 6.4.

Although Optimistic Backtracking gives rise to a substantial increase in coverage, it seems that the backtracker is not able to reach a sufficiently high accuracy to improve the sentence accuracy to a larger extent. The sentence accuracy for *ws13* using tagged preterminals (Table 7.29) is 36.2% and 36.6% in near-deterministic mode, employing ranking-based and baseline backtracking respectively, compared to 35.5% in deterministic, unrestricted parsing mode (Table 7.5).

The risk of parsing failure is dramatically reduced in CFG mode using the backtracking overlay. This would suggest that, unlike in unification mode, the backtracking constraints are not too strict to substantially improve the coverage when using CFG mode, and a relaxation of these constraints would not necessary lead to a higher *sentence accuracy*, although it is likely that the coverage would increase further if the 'optimistic' backtracking constraints were relaxed.

Conclusion for Near-Deterministic Parsing

The Optimistic Backtracking overlay introduced in Chapter 6 presents a heavily constrained backtracking strategy that requires minimal overhead and added complexity. The overlay facilitates near-deterministic parsing for a deterministic transition-based system. Two methods for selecting the backtracking points, ranking-based and baseline, are in this chapter evaluated on CuteForce.

Although the evaluation in Section 7.6.5 suggested that the ranking-based approach reached a higher accuracy than our baseline method, the parsing results were not similarly conclusive with respect to which is the better backtracking strategy in a parsing system. From the error analysis in Section 7.6.4 we did however see that the baseline method was strongly biased towards selecting a transition $c_n.t_n$ that occurred *after* the incorrect parsing decision was made in configuration c_e, where $n > e$. One of the incentives for designing a ranking-based backtracker was to utilize global syntacto-semantic information from the (partial) HPSG derivation to describe each candidate transition in a feature vector, rather then relying exclusively on local transition-specific information, which would have few means to recognize that a given transition $c_n.t_n$ has already deviated from the gold standard search path, and is therefore, for reasons unrelated to the local context of the probability distribution of the applicable transitions, not the gold standard target for backtracking. Compared to the baseline

method, we find the results for ranking-based backtracking to be encouraging, and we observe that the tendency to revert to a configuration c_n where $n > e$ is much less pronounced in the ranking-based approach, suggesting that the ranking-based mode is better equipped to disregard a configuration c_n that occurs after a parsing error in c_e. We also believe that improvements in the ranking-based approach can be obtained through further investigation and experimentation with e.g. learning algorithms and feature engineering. This distinguishes the ranking-based approach from classic approaches that are dependent on the local transition probability distributions produced by the parsing oracle.

Based on the findings in this chapter (specifically Section 7.6), we see that the near-deterministic methods are contributing to a large increase in coverage when parsing in CFG approximation mode. However, we see at best only marginal increases in sentence accuracy compared to the deterministic CuteForce configuration. This suggests that the accuracy of the backtracker, i.e. its actual success rate in repairing an incorrect transition predicted by the parsing oracle, is relatively low. Coupled with the optimistic constraints, designed to minimize computational complexity, it would seem that the search space and complexity of the backtracking task (discussed in Section 6.3.1) is too large for the backtracker to be able to repair all inaccuracies predicted by the parsing oracle for a substantial amount of sentences.

From a robustness perspective, a deterministic configuration using unrestricted parsing mode is superior to the near-deterministic approaches evaluated here. Offering a 100% coverage, the unrestricted parsing mode will also have a substantially higher semantic and syntactic F-measure compared to the near-deterministic configurations which are disposed for parsing failure. Using robust unification (see Section 7.1.2), we could also devise well-formed semantic structures from all HPSG derivations produced by the parser. However, the sentence accuracy would be slightly lower in the deterministic configuration using unrestricted parsing mode, compared to one of the configurations that facilitates backtracking. Ultimately the selection of a configuration would depend on the requirements and nature of the parsing task, and should be selected based on these concerns.

7.7 Related Parsers for ERG

In Chapter 2 we introduced the PET HPSG Parser (Callmeier, 2000, see Section 2.5.3) and Jigsaw (Zhang and Krieger, 2011, see Section 2.5.1). While both parsers produce analyses grounded in ERG, they take two different approaches to parsing, offering different trade-offs with respect to parsing precision, robustness and efficiency.

For the parsing results presented in the following, the parsers assume gold standard pretokenized test data, except for the PET parser on the out-of-

domain *cb* data set, where only the results for the untokenized data set were available. However, we observed only marginal differences between the syntactic and semantic evaluation for PET when comparing the parsing results for PET over the untokenized and (gold standard) pretokenized *ws13* (the PARSEVAL syntactic F-measure was 86.9% for untokenized data, compared to 87.1% for the pretokenized data set, as reported in Table 7.40). Hence, we do not expect that the parsing results for *cb* would have been much different if we had evaluated on a gold standard pretokenized data set for *cb*.

Note that in the syntactic evaluation, PET and Jigsaw report Exact Match (*EM*) rather than Sentence Accuracy (*SA*), which is conventionally reported by EVALB. The Exact Match metric evaluates that the parsed derivation is identical to the gold standard, while the sentence accuracy reports on the proportion of sentences where the set of constituent, and their span, is identical in the gold standard and parsed derivation.

7.7.1 PET Parser

The PET Parser opts to construct the full parse forest of analyses that are licensed by the unification-based grammar, and produces an *n*-best list based on discriminative parse ranking. Given the complexity of parsing a unification-based grammar, the PET Parser performs a very high number of computations, resulting in high parsing times compared to CuteForce. When the construction of the parse forest takes more than a preset time, in this case 1 minute, parsing fails for the given sentence. Given that the PET Parser directly employs the ERG, all derivations are guaranteed to be consistent.

The PET Parser has full coverage on the test data used in our evaluation, because only the utterances from the WeScience Corpus (introduced in Section 3.4.1) and The Cathedral and the Bazaar (*cb*, see also Section 3.5) that the PET Parser was able to parse were treebanked in the respective test sets. In itself this can be regarded an unfair advantage when comparing the parsing results to other parsers, since the PET Parser is not penalized for the subset of sentences which it was not able to parse, as these sentences are simply removed from the test data. Flickinger et al. (2010, see Section 3.4) reported that the current treebank coverage in WeScience is just below 80%, indicating that for approximately 20% of the utterances in the WeScience Corpus, PET failed to produce a parse. This should be kept in mind when considering the results for the PET Parser, presented in Tables 7.40 and 7.41.

Tables 7.40 and 7.41 reveal that the PET Parser produces more accurate analyses than CuteForce. From the parsing results for CuteForce in the various configurations, the only directly comparable parsing results are obtained when parsing with supertagged preterminals in unrestricted mode, since this configuration ensures that the set of test sentences are identical.

Data	PR	RE	F1	EM	TA	MS
ws13	87.1	87.1	87.1	48.8	96.5	2.3k
cb	84.5	84.8	84.6	26.3	94.6	4.3k

Table 7.40: Syntactic results and parsing time for *ws13* and *cb* for the PET Parser.

Data	E_A-P	E_A-R	E_A-F1	E_P-P	E_P-R	E_P-F1	E_{AP}-F1
ws13	86.6	86.4	86.5	94.1	93.9	94.0	91.5
cb	85.2	85.5	85.3	93.8	94.2	94.0	91.3

Table 7.41: Semantic evaluation for *ws13* and *cb* for the PET Parser.

Compared to the deterministic parsing results for CuteForce for this configuration, the PARSEVAL F-measures for CuteForce are 81.7% (*ws13*) and 77.4% (*cb*) (see Table 7.5), compared to 87.1% (*ws13*) and 84.6% (*cb*) for the PET Parser. In the semantic evaluation, PET has EDM_A and EDM_P F-measures of 86.5% and 94% for *ws13*, compared to 80.4% and 91.7% for CuteForce. For the semantic out-of-domain evaluation of *cb*, we see that PET has EDM_A and EDM_P F-measures of 85.3% and 94.0%, compared to 77.2% and 90.4% for CuteForce.

A metric where CuteForce is clearly better than the PET Parser is parser efficiency. With average parsing times of 12.2 *(ws13)* and 14.3 *(cb)* milliseconds per sentence in deterministic, unrestricted parsing mode, CuteForce is more than two orders of magnitude faster than PET. Further, CuteForce should theoretically reach 100% coverage for any utterance when parsing in unrestricted parsing mode. Hence, we can conclude that although CuteForce does not reach the same level of parsing precision as the PET Parser, it offers a substantial trade-off in terms of efficiency and robustness, demonstrating that a transition-based, deterministic parsing approach could be an attractive alternative to the PET Parser for use-cases where these attributes have a high priority.

7.7.2 Jigsaw

The PCFG parser developed by Zhang and Krieger (2011), Jigsaw, applies an induced PCFG approximation of the ERG. Disregarding unification, it operates in cubic time. Although the methodologies of Jigsaw and CuteForce are distinctly different, they both represent a corpus-driven approach to 'deep' linguistic processing, which is in contrast to the grammar-based methodology applied by the PET Parser, which has been the more classic approach to parsing with unification-based grammars. Since it does not perform any unification, the analyses are not guaranteed to be consistent, similar to CuteForce when operating in CFG or unrestricted mode.

The performance and efficiency of the parser are related to the size of the training data. Highest parsing quality is reached when using the entire WikiWoods (Flickinger et al., 2010, see Section 3.4.3) and two levels of grandparenting in the PCFG, in total extracting approximately 4 million CFG rules. The complexity of the PCFG does however come with a price, and the best-performing configuration of Jigsaw is in its current state approximately four orders of magnitude slower than the deterministic CuteForce configuration in unrestricted or CFG parsing mode. Zhang and Krieger (2011) present a number of configurations, applying variable amounts of training data and linguistic granularity. In Tables 7.42 and 7.43 we present the parsing results for the two best-performing configurations presented in Zhang and Krieger (2011), arranged by the training set *(TS)*.[8] *ww* represents the PCFG extracted from the entire WikiWoods, whereas *ww00* applies a subset of 482,000 sentences from WikiWoods, where approximately 320,000 CFG rules were extracted.

It should be noted that unlike the supertagger of CuteForce, Jigsaw does not convert generic lexical entries back to their native form, and is hence creating a larger search space for the task of assigning lexical entries/preterminals. The conversion of generic lexical entries is discussed in Section 4.3.2 and Appendix A.

TS	Data	PR	RE	F1	EM	TA	MS
ww	ws13	85.0	83.8	84.9	37.5	93.6	125.2k
ww	cb	79.5	79.4	79.5	18.1	89.6	397.1k
ww00	ws13	81.9	81.1	81.5	28.5	92.4	7.5k
ww00	cb	76.5	74.8	75.6	14.0	87.6	23.7k

Table 7.42: Syntactic results for Jigsaw on *ws13* and *cb*.

TS	Data	E_A-P	E_A-R	E_A-F1	E_P-P	E_P-R	E_P-F1	E_{AP}-F1
ww	ws13	84.6	83.8	84.2	92.9	92.6	92.8	89.9
ww	cb	79.9	78.1	79.0	90.6	90.7	90.6	87.0
ww00	ws13	81.6	80.0	80.8	91.2	90.0	90.6	87.3
ww00	cb	76.9	73.9	75.4	89.6	87.0	88.3	84.2

Table 7.43: Semantic results for *ws13* and *cb* for the PCFG parser in Zhang and Krieger (2011).

Compared to the deterministic configuration of CuteForce using tagged preterminals (Tables 7.3 and 7.7), we find that the *ww* configuration of Jigsaw is reaching a higher syntactic and semantic accuracy. CuteForce

[8]Parsing times are obtained through personal correspondence with the developers of Jigsaw. In its current state, Jigsaw is not optimized for parser efficiency, but the developers expect that substantial optimizations can be obtained.

obtained a PARSEVAL F-measure of 81.7% and 77.4% for *ws13* and *cb* respectively, compared to 84.9% and 79.5% for Jigsaw. The semantic evaluation consolidates this trend, and CuteForce reached an EDM_{AP} F-measure of 87.9% *(ws13)* and 86.3% *(cb)* versus 89.9% *(ws13)* and 87.0% *(cb)* for Jigsaw.

Although Jigsaw reached better parsing results than CuteForce both for the in-domain *ws13* and the out-of-domain *cb* data sets, we saw a much less pronounced difference for *cb*. A possible explanation to this may be that Jigsaw is trained exclusively on Wikipedia data, whereas some proportion of CuteForce's training data is derived from non-Wikipedia data, possibly making CuteForce more robust to other sources of data. Alternatively, there may be inherent properties with the parsing approaches of CuteForce and Jigsaw which make CuteForce better equipped to parse out-of-domain data.

For the *ww00* configuration of Jigsaw, which is trained on approximately three times the amount of training data that CuteForce uses, we see that the difference between in-domain and out-of-domain parsing scores is clearer. We find that Jigsaw's parsing scores are very similar to CuteForce's for *ws13*, with syntactic and semantics (EDM_{AP}) F-measures of 81.5%/87.3% for Jigsaw versus 81.7%/87.9% for CuteForce. For the out-of-domain set, CuteForce obtains a higher accuracy than Jigsaw, and the corresponding F-measures for the syntactic and semantic evaluations are 77.4%/86.3% for CuteForce, and 75.6%/84.2% for Jigsaw.

These results demonstrate that CuteForce, when applying approximately 33% of the amount of training data that is used in the *ww00* configuration of Jigsaw, achieves similar or higher accuracy than Jigsaw. Further, Zhang and Krieger (2011, Table 2) report an PARSEVAL F-measure of 66.45% for the *WS-PCFG(FP1)* configuration parsing *ws13*, which only applies WeScience 1-11 as training data. With the same source of training data, CuteForce obtains a PARSEVAL F-measure of 79.9% for *ws13*,[9] applying a feature model that is scaled for much larger amounts of training data. This resonates well with one of the motivations for deterministic parsing described in Nivre et al. (2007):

> Moreover, while the accuracy of a deterministic parser is normally a bit lower than what can be attained with a more complex statistical model, trained and tuned on large amounts of data, the deterministic parser will often have a much steeper learning curve, which means that it may in fact give higher accuracy with small training data sets. (Nivre et al., 2007, p. 97)

Jigsaw and CuteForce exhibit two opposing trade-offs: While CuteForce has a much higher parse efficiency than Jigsaw, Jigsaw is able to utilize the entire

[9] This score is not explicitly reported elsewhere in the thesis, but is obtained by applying only WeScience 1-11 as training data for CuteForce, used when evaluating the learning curves which were presented in Figure 7.6 and 7.7

WikiWoods as training data. A reasonable conclusion on the various trade-offs of the two corpus-driven approaches to HPSG may be that CuteForce seems better equipped to obtain a high parsing accuracy on limited amount of training data, and obtains a much higher parsing speed, whereas Jigsaw can utilize a very large amount of training data, and reach higher parse accuracy than CuteForce if a sufficient amount of data is available.

7.8 Summary and Conclusion

In this chapter we have presented the parsing results of CuteForce in all supported configurations. Further, we have compared our results with two other parsers using ERG, namely the PET HPSG Parser (see Section 7.7.1) and Jigsaw (Section 7.7.2). For the two-stage classification setup (Section 7.5) and the near-deterministic parsing configurations (Section 7.6) we previously presented an evaluation compared to the (default) deterministic configuration.

In this summary and conclusion we will therefore focus on the deterministic CuteForce configuration, based on the results we presented in Section 7.4. In Section 7.8.1 we will compare the alternative parsing modes within this scheme, and evaluate the different trade-offs these alternative modes exhibit. In Section 7.8.2 we will summarize and conclude on the trade-offs we saw between CuteForce, and the alternative ERG parsers presented in Section 7.7.

7.8.1 CuteForce Parsing Modes

CuteForce supports three different parsing modes, namely *Unrestricted mode*, *CFG approximation mode* and *Unification mode* (the parsing modes were introduced in Section 5.2.5). In theory these modes have different trade-offs – the unrestricted mode facilitates robust parsing where 100% coverage is guaranteed, and unification mode ensures that all analyses are consistent feature structures within the HPSG framework. The intention of the CFG approximation mode is to filter out implausible parser decisions through the use a CFG approximation extracted from 300,000 sentences from WikiWoods.

Unrestricted Mode The unrestricted parsing mode will execute the highest ranked *applicable* transition predicted by the parsing oracle. Whether a transition is applicable is defined by the transition system. This parsing mode ensures 100% coverage of the test data, yet, similar to the CFG approximation mode, it does not ensure that the parse derivations are unifiable HPSG analyses.

The two other parsing modes can be seen as filters to the unrestricted parsing mode, effectively blocking a transition if it violates a constraint de-

fined by the two other parsing modes. Since the two other parsing modes can lead to parsing failure, the most straightforward evaluation metrics across parsing modes are sentence accuracy (Tables 7.4 and 7.5) and consistency (Tables 7.2 and 7.3).

While there are no substantial differences between the level of consistency between unrestricted and CFG approximation mode, we see a noticeable increase up to the consistency scores reached in unification mode. Parsing *ws13* using tagged preterminals (Table 7.3), we find an error reduction of 7.6% between the consistency scores for unification mode versus unrestricted mode. For the out-of-domain *cb* test set, the corresponding error reduction is 3.1%.

Comparing the sentence accuracies, we find much smaller differences. While the sentence accuracy for *ws13* using tagged preterminals is marginally higher in unification mode (the error reduction is 0.9%, see Table 7.5), we find the opposite result for the *cb* data set – this result has occurred due to incorrect preterminals which have lead to an inconsistent feature structure, and was discussed in Section 7.4.2.

The results suggest that the design of the transition system in itself offers sufficient constraints for the classifier that predicts each transition (i.e. the oracle), and that the parsing modes that provide additional constraints to the parser do not substantially increase the overall parsing accuracy. However, since the unrestricted parsing mode does not validate the parse derivations with respect to consistency, we propose the use of robust unification in line with the method proposed by Zhang et al. (2012), this method would also be comparable to the default unification strategy proposed by Ninomiya et al. (2011) for transition-based HPSG parsing for the Enju grammar, see Section 7.2.1.

CFG Approximation Mode From a consistency perspective, we find that the CFG approximation does not contribute to more well-formed parsing derivations, compared to the unrestricted mode (see Tables 7.2 and 7.3), and nor does it contribute to a substantial increase in sentence accuracy (Tables 7.4 and 7.5) This trend is consolidated in the two-stage transition configuration in Section 7.5.

While the evaluation of Jigsaw (Section 7.7.2) attests to the validity of extracting a (P)CFG from WikiWoods, it seems that the CFG approximation applied by CuteForce did not contribute to improving the precision of the parsing oracle. Possible reasons for this may be that the CFG approximation in its current form is too naive – disregarding probabilities and invalidating all transitions which are not seen in the CFG approximation. Although we did experiment with CFG approximations extracted from much larger sources of training data, this did not contribute to higher parsing precision. Incorporating a probabilistic CFG to the transition-based

HPSG parser may possibly have more potential. We have however not conducted further studies regarding more advanced configurations and uses of a (P)CFG in relation to CuteForce.

Unification Mode The unification mode distinguishes itself from the two other parsing modes as it makes use of a third-party back-end, namely LKB (Copestake, 2002), making it distinctly slower than the other parsing modes. With a native implementation that performs destructive unification we expect however that we could achieve a substantial speed-up, considering that the current back-end is designed for non-destructive unification, a much more computationally expensive operation that is not necessary within a deterministic parsing scheme.

Ensuring that no transitions violate the constraints of the HPSG, it is the only parsing mode which guarantees that the parse derivations are consistent feature structures. This does however lead to a large number of parsing failures, compared to the other parsing modes.

Conclusion for Parsing Modes In its current state, the CFG approximation has a negligible contribution to the overall parsing quality of CuteForce. We believe that the parsing oracle in itself is so precise that the CFG approximation in its present state is too naive to further contribute to the accuracy of the parser. However, CFG approximations have successfully been implemented in other parsing systems for ERG (Zhang et al., 2012; Evensberget, 2011; Cramer, 2011) and Enju (Matsuzaki et al., 2007, inter alios), and we do believe that further investigation and development of CFG filtering could contribute to higher precision for a transition-based system for ERG. We have however not carried out further studies within this field.

Through unification mode the proportion of inconsistent parse derivations decreased with 7.6% (compared to unrestricted and CFG approximation mode) for *ws13* and 3.1% (compared to unrestricted mode)/4.6% (compared to CFG approximation mode) for *cb*, (see Table 7.3). However, this increase in consistency may for many applications not justify the added computational cost of performing unification. Alternatively, we could opt for using unrestricted parsing mode, and through the use of robust unification overwrite grammatical inconsistencies, or filter out inconsistent derivations if default unification is not desirable for a certain use-case. The use of default unification is now well established for data-driven approaches to unification-based parsing (Zhang et al., 2012; Ninomiya et al., 2011, inter alios), and for most use-cases we find this to be a very attractive trade-off which would hence combine the efficiency and robustness of the unrestricted parsing mode and simultaneously ensure that all utterances in the test data receives a semantic representation. The running time for an application performing robust unification is however not included in the parsing times

which are accounted for in the presentation of the deterministic and near-deterministic configurations of CuteForce presented in this chapter.

7.8.2 CuteForce and Alternative ERG Parsers

Numerous research initiatives have been devoted to advancing deterministic and near-deterministic transition-based parsing within a range of grammatical frameworks. CuteForce is to our knowledge the first initiative to explore transition-based parsing for the English Resource Grammar. We have demonstrated a system capable of achieving a high level of accuracy with very attractive parsing times, facilitating use-cases were parsing efficiency is crucial. Transition-based approaches can in principle also facilitate word-by-word processing, as opposed to traditional sentence-level parsing typically associated with non-deterministic chart parsing approaches.

CuteForce does not achieve the same level of accuracy as the best-performing configurations of the non-deterministic ERG parsers discussed in Section 7.7. Compared to PET and Jigsaw, we see a clear trade-off between parsing efficiency and accuracy, where CuteForce is approximately two and four orders of magnitude faster than the aforementioned non-deterministic parsers. However, it is interesting to observe that CuteForce seems to parse more accurately than Jigsaw when the amount of training data is limited. The evaluation of the *ww00* configuration of Jigsaw (see Section 7.7.2) suggests that, despite being trained on approximately three times the amount of training data, CuteForce is achieving similar or higher parsing accuracies than Jigsaw. We find these results encouraging, affirming that a transition-based, deterministic parsing approach can be competitive with the non-deterministic corpus-driven approach developed by Zhang and Krieger (2011) also in terms of accuracy.

7.8.3 Summary

The findings presented in this chapter suggest that deterministic, transition-based parsing is a viable parsing strategy for a large-scale unification-based precision grammar. Although our parser did not obtain state-of-the-art parsing accuracy, it reached a high level of accuracy, and proved substantially faster than the alternative ERG parsers we have evaluated. CuteForce provides a corpus-driven approach that enables transition-based parsing for a unification-based grammar. We assume that our approach can be adapted to other frameworks, given that a sufficient amount of annotated training data is available. By facilitating the use of alternative parsing configurations and robust unification, we have demonstrated that transition-based, deterministic parsing offers a range of attractive trade-offs, making it a feasible alternative to other, non-deterministic parsers for HPSG.

Chapter 8

Summary and Concluding Remarks

In this thesis our aim has been to investigate efficient, data-driven deterministic transition-based parsing for a large-scale, unification-based precision grammar. Specifically, we have designed and implemented a parsing platform targeting deterministic and near-deterministic processing of the DELPH-IN grammars, which are a multi-lingual family of hand-crafted deep grammars for HPSG. We have optimized our parsing platform for the English Resource Grammar (ERG), the by far most comprehensive of the DELPH-IN grammars.

Over the recent decade there have been a number of research initiatives targeting deterministic and near-deterministic, transition-based parsing approaches. These efforts have led to the development of parsing systems that have obtained competitive parsing results, or even state-of-the-art accuracy within grammatical frameworks such as dependency grammar (Zhang and Nivre, 2011; Nivre et al., 2007), CFG (Sagae and Lavie, 2005), CCG (Zhang and Clark, 2011a), inter alia.

Deep, hand-crafted semantic grammars have not to the same extent been targeted by research initiatives using deterministic and near-deterministic parsing schemes. Semantically rich grammars constitute a large search-space because they opt to make a number of fine-grained semantic distinctions. In this light, deterministic processing may be both a strength and an Achilles heel for a parsing platform. Deterministic parsing is arguably the most computationally efficient parsing approach as it spends a minimal amount of time on pursuing analyses which will not be a part of the final derivation, typically offering a linear time-complexity. However, given the increased linguistic granularity provided by the DELPH-IN grammars, it was not self-evident that the trade-offs observed for other formalisms would hold in a semantically richer framework. Although we have very recently seen other data-driven approaches to HPSG parsing for the ERG framework, this thesis

is to our knowledge the first research initiative that explores deterministic transition-based parsing for this grammar.

8.1 Summary of the Components of the Project

In the Introduction in Chapter 1, we defined four contributions, i.e. four separate tasks which we sought to address in the PhD project. In the following, we will give a brief summary of these four components, and conclude on the findings our investigation has yielded.

Augmenting Linguistic Resources

The primary goal for this thesis has been to explore deterministic transition-based parsing for DELPH-IN style grammars. For semantically less expressive grammatical frameworks, like CFG and dependency grammar, data-driven approaches have assumed a dominant position. Data-driven parsers typically require a sizable amount of training data. While the Wall Street Journal section of Penn Treebank (Marcus et al., 1993) employed in a range of projects targeting CFG parsing, dependency parsing and even HPSG parsing (this specifically concerns the Enju grammar, introduced in Section 2.5.4), consists of over 1 million tokens, the Redwoods Treebank, annotated with the ERG, held approximately 220,000 tokens at the beginning of this project, drawn from a number of different domains. Acknowledging that this amount of data was simply too little to develop a competitive statistical parsing platform, the first stage of the PhD project was devoted to augmenting the linguistic resources of the ERG.

With the release of the WeScience Treebank and WikiWoods (Ytrestøl et al., 2009; Flickinger et al., 2010), the RedWoods Treebank has been augmented with approximately 130,000 tokens. Additionally, WikiWoods consists of 47 million automatically annotated sentences, facilitating data-intense research targeting the ERG to a range which was unprecedented prior to these releases. This has already manifested itself in two initiatives targeting corpus-driven PCFG approximation of the ERG (Zhang and Krieger, 2011; Evensberget, 2011), projects that directly depend on the new linguistic resources made available over the course of this project. Further, these resources have been crucial for the remaining three components of our project, which we will address below.

Supertagging

Chapter 4 presents a supertagger that assigns preterminal lexical types within the ERG framework. In the training of the supertagger we adapted and evaluated two off-the-shelf taggers, namely the C&C supertagger (Clark and Curran, 2007) and SVM^{hmm} (Joachims et al., 2009). While the SVM^{hmm}

tagger achieved a high level of accuracy when the training data was limited, the C&C supertagger distinguished itself as being much more scalable.

Taking advantage of the extensive linguistic resources made available earlier in the project, we applied 10 million training sentences (142 million tokens) from WikiWoods in our best-performing configuration, thus reaching a previously unmatched level of supertagging accuracy for the ERG. That these results were obtained using training data that consisted solely of automatically annotated data, is an insight which is encouraging for other projects that suffer from a limited amount of gold standard annotations, attesting that a high level of supertagging accuracy can be obtain even without the use of gold standard training data.

Deterministic, Transition-Based HPSG parsing

In Chapter 5 our transition-based, deterministic HPSG parser, CuteForce, is presented. This is to our knowledge the first parser that applies a deterministic parsing approach to the ERG. Similarly to the supertagger, CuteForce relies heavily on the linguistic resources that were developed during this project. Optimized for Wikipedia, CuteForce is trained on the Redwoods Treebank and WikiWoods. Although the gold standard Redwoods Treebank was included in the training data, our findings suggest that a parser trained strictly on automatically annotated data reach an only marginally lower accuracy (see Section 7.3).

While alternative use-cases may have different priorities, we find that the deterministic configuration operating in unrestricted parsing mode offers the best trade-off between parsing time and accuracy. This configuration enables robust parsing with 100% coverage. Since the unrestricted parsing mode does not perform grammatical constraint checks during parsing, the final parse derivations may contain inconsistencies in the feature structure, and we suggest *Robust Unification* (see Section 7.1.2) to devise approximative semantic representation from these inconsistent analyses.

Although CuteForce was not able to reach the state-of-the-art accuracy obtained by non-deterministic parsers, we find the results encouraging. Compared to the corpus-driven HPSG parser developed by Zhang and Krieger (2011), our findings suggest that the deterministic CuteForce configuration can provide better parsing scores when the amount of training data is limited (and specifically, limited to what CuteForce can utilize for training).

By this we have demonstrated that deterministic transition-based processing is a viable parsing strategy for a large-scale unification-based precision grammar like the ERG. This resonates well with the findings in Ninomiya et al. (2011), developed partially parallel to our own project, where the authors evaluated the feasibility of employing a deterministic shift-reduce parsing scheme for the Enju grammar.

Near-Deterministic Backtracking Overlay

Chapter 6 presents a near-deterministic overlay that enables moderate backtracking for a deterministic transition-based parser. Its main objective is to identify and repair an incorrect parsing decision if the deterministic parsing ends with failure due to hard constraints in the grammar. The backtracking approach, dubbed *Optimistic Backtracking*, defines a heavily constrained framework that entails a minimal amount of overhead in terms of complexity and implementation to a deterministic parsing platform, and our studies showed that the parsing times are only up to 150% higher when applying this near-deterministic overlay, compared to the deterministic configuration.

While conventional backtracking strategies typically only consider the probability distribution in the list of valid transitions for each transition (see Section 6.4.3), we have developed an innovative ranking-based backtracking approach which considers the sequence of transitions and the partially built HPSG derivation when selecting a backtracking point. The independence from explicit transition probabilities could be vital for backtracking approaches devised for transition-based parsers which do not provide such probability scores.

Experimentations carried out on CuteForce reveal that the ranking-based approach in isolation consistently reached higher accuracies than the 'baseline' approach (see Section 6.7) in terms of identifying the correct point for backtracking, attesting that the use of ranking may contribute to improving the accuracy of a backtracker whose goal is to identify and repair an incorrect parsing decision.

In our evaluation of near-deterministic parsing (see Section 7.6) we saw that the sentence accuracy improved only marginally, compared to the corresponding deterministic configurations. It did however contribute to a substantial increase in coverage in some configurations. Much more so than conventional backtracking methods, ranking-based backtracking offers a range of options for customization. We believe that a higher parser accuracy could be obtained through further experimentation and refinement of the ranking-based approach, and possibly also through a relaxation of the backtracking constraints which are used in this current configuration.

8.2 Conclusion for our Research Questions

In the introduction to this thesis, we defined four research questions to which we have sought an answer in this project. We will repeat these questions here:

1. Can transition-based, deterministic parsing be scaled up to a broad-coverage precision grammar like the English Resource Grammar?

2. Which type of linguistic resources are required in order to enable a data-driven deterministic parsing system for this kind of grammatical formalism, and to which scale?
3. To what degree do techniques and experiences applied in other frameworks transfer to this setup?
4. What trade-offs (e.g. between parser precision, robustness, and efficiency) can be observed?

In the following paragraphs, we will provide our conclusion to these research questions.

Feasibility of the Method While we have seen deterministic parsers reaching state-of-the-art accuracy for dependency parsing, it was not self-evident that this was a feasible parsing strategy for a deep semantic grammar. Our parsing results indicate that although we have not been able to reach the same level of accuracy as the PET parser, which is the 'standard' parser for the ERG and the larger DELPH-IN community, our parsing system reaches a high level of accuracy, coupled with very attractive parsing times.

Using unrestricted parsing mode, CuteForce offers 100% coverage with parsing times averaging from 12-17 milliseconds per sentence. This is, to our knowledge, the fastest reported parsing times for the ERG, and demonstrates that transition-based deterministic approaches can facilitate parsing for a range of applications and use-cases where conventional non-deterministic chart parsers would be too slow.

Linguistic Resources Over the course of this project, we have substantially increased the linguistic resources for ERG. The linguistic resources have been used in three different components of the parsing platform, namely in training the supertagger, the parsing oracle and the ranking-based backtracker. In the current setup, the supertagger is by far the heaviest consumer of linguistic resources, using more than 20% of the sentences in WikiWoods in the training of the best-performing configuration. In our evaluation in Chapter 7 (consider the learning curves presented in Section 4.4), it seems clear that without substantially augmenting the linguistic resources of ERG, our supertagger would be performing on a much lower level. This would have had direct consequences on the parsing accuracy.

We saw that the parsing oracle reached a relatively high level of accuracy when the amount for training data was limited, compared to the PCFG parser introduced by Zhang and Krieger (2011), and this was discussed in Section 7.7.2. However, CuteForce clearly benefited from more training data, as can be observed in the learning curves in Figures 7.6 and 7.7. In the best-performing configuration, we applied 150,000 sentences, effectively

using the entire Redwoods Treebank, and 120,000 additional sentences from WikiWoods.

Further, the ranking-based backtracker depended on training data that was derived from parsing of sentences that was not previously used when training the parsing oracle. In practice this implied using additional training data from WikiWoods, and in the current configuration we used up to 150,000 WikiWoods sentences in the training of the ranking-based backtracker.

It is hard to define the scale of linguistic resources that is *required* to implement and assess a competitive deterministic parsing platform for a DELPH-IN style grammar, because this would depend on the level of accuracy that is demanded. In our parsing pipeline, the supertagger is by far the heaviest consumer of training data. Using less training data would result in a worse-performing supertagger, which would propagate inaccuracy to the subsequent parsing stage. From our design of the parsing platform, it is however clear that the scale of the linguistic resources that existed prior to this project would have been insufficient to train a parsing system that would be able to reach a satisfactory level of accuracy for most use-cases.

Transfer of Techniques and Experiences From Other Frameworks
In the development of the deterministic transition-based parsing platform, we have been inspired by a number of research initiatives. We have opted for a transition setup that applies a greedy local best-first search in a deterministic configuration, a strategy which is also investigated in Ninomiya et al. (2011); Nivre et al. (2007); Sagae and Lavie (2005), inter alios. In its current state, the transition-system employed by CuteForce facilitates unary and binary projections in a shift-reduce fashion, similar to the CFG parser developed by Sagae and Lavie (2005). However, compared to CFG and dependency parsing, the hard constraints of HPSG may cause inconsistencies in the parse derivations, resulting in parse failures. This is addressed in Ninomiya et al. (2011), where they propose default unification to overwrite grammatical inconsistencies and ensure that the parse analyses represents unifiable feature structures. However, contrary to Ninomiya et al. (2011), we have evaluated a parsing strategy which does not employ unification when selecting parser actions. Our results show that this method, combined with robust unification applied in a postprocessing stage, is a competitive parsing strategy which can achieve very attractive parsing times.

We were positively surprised by the very small reduction in precision when omitting the costly unification operation. The feasibility of training a parsing oracle which reaches a high degree of accuracy for a unification-based grammar without applying an expensive unification operation is, the way we see it, very encouraging, enabling parsing for large-scale precision grammars at parsing speeds that are typically associated with much shal-

lower grammatical frameworks.

Trade-offs Compared to other Frameworks In Chapter 7, Section 7.7 we compared CuteForce to two other parsers for ERG, namely Jigsaw (Zhang and Krieger, 2011) and the PET HPSG Parser (Callmeier, 2000). As opposed to PET, both CuteForce and Jigsaw represent data-driven approaches that can more easily be ported to new languages or grammatical frameworks, provided that the linguistic resources were available. CuteForce on the other hand distinguishes itself from PET and Jigsaw as it is makes use of a transition-based parsing approach, while PET and Jigsaw use a chart parsing scheme.

In terms of parser accuracy, we see that CuteForce in its current state is not able to reach the same level of accuracy as the two chart parsers, where the PET parser achieves the highest parsing accuracy both for the in-domain and out-of-domain test sets. The best-performing configuration of Jigsaw also reaches a higher level of accuracy than CuteForce, and we see that the differences are more distinct for the in-domain data set, compared to out-of-domain data (see Section 7.7.2). With the current feature model, we were able to use up to 150,000 sentences when training the parsing oracle for CuteForce. Jigsaw was on the other hand able to utilize the entire WikiWoods in its language model. However, when Jigsaw's training data was limited (i.e. limited to what CuteForce was able to utilize), CuteForce was able to reach a higher level of parsing accuracy than what was reported for Jigsaw in Zhang and Krieger (2011).

While the PET parser and Jigsaw on average requires seconds and even minutes of processing time per sentence,[1] the deterministic configuration of CuteForce uses in the range of 11-17 milliseconds per sentence, excluding the unification mode and the robust unification post-processing stage. This could enable the use of deep semantic parsing, and specifically the use of ERG, in areas where existing parsing systems prove to be too inefficient. In the current version, CuteForce does however require pretokenized input, and cannot be used to parse natural running text. This will be discussed in Section 8.4.4, where we propose a feasible approach that could resolve this issue.

Compared to Jigsaw and PET, which are both instantiations of chart parsers, another potential trade-off with deterministic transition-based parsing is that it facilitates the use of production-time in processing, an aspect which becomes increasingly important following the accelerated interest for human-computer-interaction. However, to enable this, the transition-based parsing platform would have to support input stream processing. In its current version, this is not supported by CuteForce, and this will be further discussed in Section 8.4.4.

[1] It should be repeated that Jigsaw is in its current state not optimized for efficiency.

8.3 Availability of Resources

In this project we have developed a wide range of resources which we have made, or aim to make, publicly available. In this section we will discuss these resources, and provide details on how they can be acquired.

Corpus Resources

The WeScience Treebank and WikiWoods have been released during this project. In addition to these linguistic resources, we have also developed tools for extracting a corpus from Wikipedia. The method is further described in a technical report (Ytrestøl, 2009), and on the WeScience homepage: `http://moin.delph-in.net/WeScience`. The approach for corpus extraction has later been used to obtain new linguistic resources, and this is further discussed in Read et al. (2012), where these tools are used to extract a Wikipedia-derived corpus targeting Linux-related articles. The tools for harvesting and extracting a corpus from Wikipedia are bundled with the LOGON infrastructure, and can be accessed from `http://moin.delph-in.net/LogonTop`.

Supertagger

As we have used off-the-shelf supertaggers in this project, we have trained models for the C&C supertagger and SVM^{hmm} tagger optimized for the Wikipedia and WeScience domain. These models can, upon request to the author, be made available. Tools that are used to adapt treebank data to a valid input format for the supertaggers are bundled with the CuteForce Platform, which we will discuss below.

CuteForce Platform

The transition-based parser is implemented in Java 1.6. This parsing platform also includes a preprocessing pipeline which assumes DELPH-IN style treebanks, and produces the required input data for the C&C supertagger and SVM^{hmm} tagger, tools for converting supertagger output to the applicable CuteForce input format, and tools for validating the accuracy of the supertagger. The parser supports alternative outputs, such as the conventional phrase structure representation used in the DELPH-IN treebanks, and a reduced syntactic representation which can be used as input to EVALB, thus facilitating a syntactic evaluation with PARSEVAL.

The CuteForce Platform has been developed as an instrument for experimentation and evaluation of a large number of alternative strategies for transition-based parsing for DELPH-IN grammars. In it current state, it is not a mature software, consisting largely of undocumented code. Although it is not currently accessible from a specific download site, it can be made

available under an open source license upon request to the author of this thesis.

8.4 Further Research

Throughout the thesis, we have been forced to prioritize certain aspects, leaving a number of research questions open. Although by no means exhaustive, we here present a list of areas where we think further experimentation would be interesting.

8.4.1 Native Unification

Acknowledging that unification is a costly operation, this project has first and foremost explored the feasibility of HPSG parsing without the use of unification. By omitting unification when restricting the search-space of the oracle, we have designed a system that to our knowledge achieves the fastest parsing times reported by any parsing platform using the ERG.

In this project we have evaluated a unification-filter in one of the parsing modes of CuteForce. However, this unification operation applies the third-party LKB platform and is not optimized for deterministic parsing. By implementing a native unification-check of two feature structures, we assume that a speed-up by more than one order of magnitude could have been obtained. This could have facilitated the pursuit of alternative strategies for transition-based parsing where a broader use of normal unification would not have been prohibitively expensive.

8.4.2 Beam Search

Backtracking and beam search seem to be the most popular near-deterministic strategies for transition-based parsers. We presented these *stack search* strategies in Section 2.6.1. While we have investigated moderate backtracking, we have not evaluated beam search as a parsing strategy. In a transition-based near-deterministic parsing scheme, beam search can be attractive for a number of reasons. By adjusting the size of the beam one could flexibly experiment with configurations where various proportions of the search space are explored – keeping a small beam width would result in a smaller proportion of the search space being explored, whereas a larger beam would increase the time/space complexity. Although beam search was not pursued in this thesis, we believe, based on experiences from similar projects (most notably Ninomiya et al., 2011), that beam search could be a strategy that might have the potential to further increase the accuracy of our system.

8.4.3 Multitagging

In Section 5.2.1 we discussed the use of multitagging as a means to improve the sentence recall for the supertagger, and the effects this would have on the overall parsing platform. This strategy is pursued in a range of transition-based parsing systems, such as Zhang and Clark (2011a, inter alios), and could be accomplished by introducing an additional parser operation that would select a preterminal from the set of candidates assigned by the supertagger. However, this alternative design could potentially have negative effects for the training of the parsing oracle, since an input buffer with multiple tags per token could have substantially increased the complexity of the feature model. Further aspects of using multitagging in a transition-based parsing system were discussed in Section 5.2.1, but to date remains unexplored for the CuteForce platform.

8.4.4 Integrate Tokenization

In its current form, the supertagger presented in Chapter 4 assumes gold standard ERG tokenization. While assuming pre-tokenized input has been a common strategy for a wide range of parsing initiatives (Zhang and Krieger, 2011; Ninomiya et al., 2011; Nivre et al., 2007, inter alios), this strategy does presuppose an additional preprocessor stage before 'real-world' parsing.

While a number of tokenizers conforming to the Penn Treebank have been developed and made available, the tokenization of ERG is, mainly due to its Multiword Expression (MWE) handling, more complex than a corresponding preprocessing stage to produce Penn Treebank tokenization. Dridan (2009) presented a pipeline that processes running text and creates ERG supertags, but without explicitly tokenizing the input text to conform to the ERG. As of today, there are to our knowledge no off-the-shelf preprocessors that can convert running text to ERG tokenization.

One feasible approach to deterministic processing of untokenized text for the ERG framework would be to introduce additional parser operations that would read input from a stream, recognize word boundaries and successively identify MWEs, hence deriving and processing the ERG tokens on the fly as the input stream is parsed. This strategy could make the parser independent of a preprocessing stage, hence facilitating input stream processing on a word-by-word basis. Such modifications would however be nontrivial, and for the scope of this thesis, we have not carried out any investigation directly related to input stream processing of running, untokenized text.

8.4.5 Combination of Parsing Systems and Parsing Modes

Integrating alternative parsing strategies and a combination of parsing systems have been used successfully to boost the accuracy in many different research initiatives. Strategies for system integration includes *voting*, where

typically the majority output of n parsing systems is chosen (Sagae and Lavie, 2006; Zeman and Žabokrtský, 2005), or *stacking*, where the output of one parser is given as input for another (Nivre and McDonald, 2008; Martins et al., 2008).

In addition to combining various parsing systems, one could also imagine combining alternative parser configurations. As demonstrated in Chapters 5 and 6, CuteForce supports a wide range of configurations, both in deterministic and near-deterministic mode. In this thesis, we have evaluated these alternative setups individually. However, the wide range of alternative setups licenses a number of plausible strategies for combining two or more parsing strategies. As we mentioned in Section 7.6, a feasible approach that would combine the advantages of the CFG and unification parsing modes with the robustness of unrestricted parsing mode, would be to use unrestricted parsing mode as a fallback strategy, resorting to choosing the highest ranked transition ranked by the parsing oracle each time there are no transitions that are licensed in CFG or unification mode.

We assume that some optimizations could be obtained by combining different parsing systems and internal alternative parsing setups, and we regard this as an interesting subject for further study. For the scope of this thesis, we have however opted to evaluate the alternative CuteForce setups individually, as presented in Chapter 7.

Appendix A

Generic Lexical Types

In the ERG treebanks and parsed corpus (essentially Redwoods Treebank and WikiWoods), each token is assigned a lexical entry, from which the lexical types are determined. Given that the lexical entries are derived from the lexicon, unknown words are assigned generic lexical entries, and subsequently generic lexical types are determined. However, to avoid the supertagger needing to learn which words are known and unknown in the ERG, we have opted to convert the generic tags back to their native form. This mapping was carried out in consultation with the ERG developers, and is presented in Table A.1.

Certain generic types are mapped to two native tags, implying that the generic type assumes the role of a native lexical type, and a lexical rule. In Table A.1 this implies for example that *n_-_c-pl-gen_le*, holds the semantic information contained in the lexical type n_-_mc_le (mass noun), and the lexical rule *n_pl_olr* (plural). When this mapping is carried out in preprocessing, the treebank representation is rewritten, as illustrated in Figure A.1.

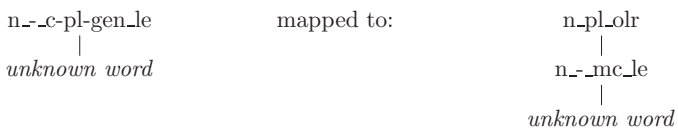

Figure A.1: Generic lexical types are mapped to corresponding native tag(s).

Generic Lexical Type	Mapped to
aj_-_i-cmp-unk_le	aj_pp_i-cmp_le
aj_-_i-crd-gen_le	aj_-_i-crd-two_le
aj_-_i-crd-unk_le	aj_-_i-crd-two_le
aj_-_i-frct-gen_le	aj_-_i-frct_le
aj_-_i-ord-gen_le	aj_-_i-ord-two_le
aj_-_i-sup-unk_le	aj_-_i-sup_le
aj_-_i-unk_le	aj_-_i_le
av_-_i-unk_le	av_-_i-vp_le
n_-_mc-ns-g_le	n_-_mc-ns_le
aj_-_i-crd-unk_le	aj_-_i-crd-two_le
n_-_c-pl-gen_le	n_-_mc_le&n_pl_olr
n_-_day-crd-gen_le	n_-_c-day_le
n_-_mc-ns-g_le	n_-_mc-ns_le
n_-_mc-unk_le	n_-_mc_le
n_-_meas-n-gen_le	n_-_c-meas_le
n_-_pn-dom-e-gen_le	n_-_pn-dom-euro_le
n_-_pn-dom-gen_le	n_-_pn-dom-card_le
n_-_pn-dom-o-gen_le	n_-_pn-dom-ord_le
n_-_pn-gen_le	n_-_pn_le
n_-_pn-unk_le	n_-_pn_le
n_-_pn-pl-unk_le	n_-_pn-pl_le
n_np_pn-hour-gen_le	n_-_pn-hour_le
n_-_c-pl-gen_le	n_-_mc_le&n_pl_olr
n_-_c-pl-unk_le	n_-_mc_le&n_pl_olr
v_-_pas-unk_le	v_-_psv_le
v_np*_pa-unk_le	v_np*_le&v_pst_olr
v_np*_bse-unk_le	v_np*_le&v_n3s-bse_ilr
v_np*_pa-unk_le	v_np*_le&v_pst_olr
v_np*_pr-3s-unk_le	v_np*_le&v_3s-fin_olr
v_np*_pr-n3s-unk_le	v_np*_le&v_n3s-bse_ilr
v_np*_prp-unk_le	v_np*_le&v_prp_olr
v_np*_psp-unk_le	v_np*_le&v_psp_olr
v_np*_unk_le	v_np*_le
generic_dom_card_ne	n_-_pn-dom-card_le
generic_card_np_ne	aj_np_i-crd-nsp_le
generic_dom_euro_ne	n_-_pn-dom-euro_le

Table A.1: Mapping of generic lexical types to corresponding native lexical types.

Appendix B

Transition Statistics for *cb*

In the parser evaluation in Chapter 7, we evaluated the over- and undergeneration for the two alternative parsing oracle configurations (single-stage and two-stage). The statistics presented in Chapter 7 are derived from parsing *ws13*. To augment the statistical basis for our conclusion, we present the corresponding transition statistics for the out-of-domain test data (the data sets used in this thesis were presented in Section 3.5), namely *The Cathedral and the Bazaar* (*cb*), using supertagged preterminals.

B.1 Transition Accuracy and Over- and Undergeneration

In Chapter 7 we opted against the use of a two-stage classification setup, since this oracle configuration would be more computationally expensive, without offering an improved parsing accuracy. We also discussed the oracle setup's dispositions for over- and undergeneration of the CuteForce transitions, where we found that even though the training data is much more balanced when using the two-stage classification setup, we saw no indication that this would result in less over- or undergeneration of the various prediction classes, i.e. the transitions.

As we in Chapter 7 only reported transition statistics for *ws13*, we will in the following augment the statistics basis for our conclusion regarding the alternative oracle classification setups by providing the similar statistics for the out-of-domain *cb* data set. We refer to Section 7.4 and Section 7.5 for the discussion regarding the two alternative oracle configurations.

B.1.1 Single-Stage Classification

In Tables B.1 and B.2 we present the transition accuracy and degree of over- and undergeneration for the single-stage classification setup for CuteForce parsing the out-of-domain *cb* data set (supertagged preterminals).

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	3774	90	87	0
UNIT	54	3158 (129)	17	0
PASSIVE	65	31	1787 (21)	0
ACCEPT	0	0	0	70 (5)

Table B.1: Confusion matrix for *cb* using supertagged preterminals, deterministic single-stage classification setup, unrestricted parsing mode.

	OG	UG
DETACH	4.5	3.0
UNIT	2.2	3.7
PASSIVE	5.1	5.5
ACCEPT	0	0

Table B.2: The degree of over- and undergeneration for the prediction of the transitions in Table B.1, i.e. parsing of *cb* using supertagged preterminals, deterministic single-stage classification setup, unrestricted parsing mode.

B.1.2 Two-Stage Classification

Tables B.3 and B.4 present the transition accuracy and degree of over- and undergeneration for the two-stage classification setup for CuteForce parsing the out-of-domain *cb* data set (supertagged preterminals).

	DETACH	**UNIT**	**PASSIVE**	**ACCEPT**
DETACH	3726	90	90	0
UNIT	53	3122 (129)	21	0
PASSIVE	63	26	1745 (24)	0
ACCEPT	0	0	0	71 (2)

Table B.3: Confusion matrix for *cb* using supertagged preterminals, deterministic two-stage classification setup, unrestricted parsing mode.

	OG	UG
DETACH	4.7	3.0
UNIT	2.3	3.6
PASSIVE	4.9	6.1
ACCEPT	0	0

Table B.4: The degree of over- and undergeneration for the prediction of the transitions in Table B.3, i.e. parsing of *cb* using supertagged preterminals, deterministic two-stage classification setup, unrestricted parsing mode.

Appendix C

CuteForce and Shift-Reduce Parsing

In Chapter 5 we presented the transition system and data structure for CuteForce. For the current transition system and data structure, a Shift-Reduce framework would have sufficed to facilitate the parsing operations we introduced in Chapter 5, and this will be demonstrated in Section C.1 below. We have however opted to use an alternative layout which enables increased flexibility, and the motivation for this will be elaborated in Section C.2.

C.1 CuteForce as a Shift-Reduce Parser

In this section we will explain how the current CuteForce data structure and transition system relate to a traditional Shift-Reduce parser, and account for the parser operations and stacks which would need to be redefined in order to reduce CuteForce to a conventional SR parser. By providing this presentation, we hope to make the overall design of CuteForce more transparent for the reader.

Shift-Reduce Data Structure The conventional Shift-Reduce framework assumes a stack S of tokens where grammatical categories (e.g. lexical types and HPSG rules) are pushed and popped, and a buffer β, holding the tokenized input sentence and possibly their preterminals. In CuteForce the S stack is divided between the δ-stack, and π_0, aka the top element of π. The top element of S, S_0, would be π_0, $S_1{=}\delta_0$, $S_2{=}\delta_1$, etc. The rest of the π stack in CuteForce, excluding the detached edges which also appear on the δ stack, contains the attached edges that are ancestors of π_0. These items are not explicitly stored in the stack within the Shift-Reduce framework, since the ancestor relations are captured within the S_0 element.

In addition to the modified data structure, the parser actions will be redefined, as presented in the following.

Shift-Reduce Transition System The transition system of CuteForce is in its current form interchangeable with a conventional Shift-Reduce transition system adapted for a maximally binary phrase-structure grammar, without any loss of functionality. Given the data structure presented above, we will in the following outline a hypothetical shift-reduce transition system which would use the same number of transitions to produce an equivalent parse derivation of an input sentence as the default CuteForce transition system we presented earlier.

- SHIFT: The shift operation would push the next preterminal from the buffer onto the S stack. This operation is parallel to the DETACH operation for CuteForce, and would exhibit the same constraint as the DETACH operation, i.e. it would only be valid if there are unprocessed elements on the buffer.
- REDUCE-UNARY(R^1): This transition would replace the UNIT transition, and pop the first element in S_0. The unary rule R^1 is pushed to S, assuming the popped item as its unary daughter.
- REDUCE-BINARY(R^2): This transition would replace the PASSIVE transition. S_1 and S_0 would be popped, as these items would be the left and right daughter of a new binary rule (R^2) which would be pushed to S. This transition would only be valid if S contained two elements or more.
- ACCEPT(s): The ACCEPT transition assigns the start symbol s and terminates the parse. This transition would only be valid if there was exactly one element on S, and the entire buffer β is consumed. S_0 would then be the root node to the parse derivation for the input buffer β.

The four transitions and the corresponding data structure proposed here are analogous to a conventional Shift-Reduce parser for a maximally binary phrase structure grammar, and would operate equivalently to the current configuration, i.e. data structure and transition system of CuteForce. We will further discuss this relation in Section C.2, especially regarding the limitations of a Shift-Reduce scheme, compared to the increased flexibility that our parsing system permits.

C.2 Motivations for the Current Design

As we discussed in Section 5.2.3, the current parsing algorithm applied by CuteForce could have been implemented as a traditional Shift-Reduce

parser. We have however opted for an alternative data structure and transition system, which we find is better scaled to facilitate customizations to the parsing algorithm, whilst maintaining a data structure that ensures that our definition of the feature functions (introduced in Section 5.3.3) would largely remain constant, although the transition system in itself would change. To exemplify, we will in the following outline two such plausible augmentations for CuteForce.

Early versus Late Commitment The current DETACH transition assumes a so-called *late commitment* strategy, where the binary rule which will be the mother of δ_0 will not be predicted until the siblings of δ_0 are found. Alternatively, an early commitment strategy would predict the mother category as soon as the first daughter (typically referred to as the *left corner* of the production rule) is processed, effectively creating what resembles an active edge in classic chart parsing.

Although the transition system would have had to undergo some modifications to enable early commitment, this would entail only small changes to the data structure. Explicitly, compared to the current data structure where the δ stack is a stack of pointers to left-corner daughters in the π stack, so-called 'detached edges' (see Section 5.2.2), the δ stack would in an early commitment scheme be a stack of unfinished ('active') edges where at least one daughter is instantiated, while others remain to be found. The instantiated daughter(s) would be similar to the pointer(s) in the current δ stack, which in its turn would ensure that the feature functions of the current data structure could be easily adapted to an early commitment scheme.

Higher Rule Arities In our discussion so far, and in parsing experiments with the ERG, we have assumed that all grammar rules are either unary or binary. However, by reserving the δ stack as a stack of 'active' edges where not all of the daughters are resolved, we could design a simple transition-based framework that facilitates parsing with grammars of higher rule arities than the grammar we have assumed for the final transition system of CuteForce. This option could in principle be maintained both within an early or late commitment scheme. For ternary grammars or higher, an additional transition could be introduced, which could attach one additional daughter at the time to an incomplete edge on δ_0, leaving the π stack unaffected.

Higher rule arities would imply that the feature functions referring to detached edges would have to be slightly redefined. We do however think that the current data structure of CuteForce, and especially the definition of the two different stacks (the π and δ stacks), makes CuteForce more robust for experimentation with alternative parsing strategies, whilst maintaining equivalent access to the features as defined in the current feature model for CuteForce (Section 5.3.6).

Bibliography

Srinivas Bangalore and Aravind K. Joshi. Supertagging: An approach to almost parsing. *Computational Linguistics*, pages 237–265, 1999.

Niels Beuck, Arne Køhn, and Wolfgang Menzel. Decision strategies for incremental POS tagging. In *Proceedings of the 18th Nordic Conference of Computational Linguistics*, pages 26–33, Riga, Latvia, 2011a.

Niels Beuck, Arne Køhn, and Wolfgang Menzel. Incremental parsing and the evaluation of partial dependency analyses. In *Proceedings of the International Conference on Dependency Linguistics*, pages 290–299, Barcelona, Spain, 2011b.

Ezra Black, Steven Abney, Daniel P. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the 4th DARPA Speech and Natural Language Workshop*, pages 306–311, Pacific Grove, USA, 1991.

Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pages 31–37, Philadelphia, USA, 1992.

Phil Blunsom. *Structured Classification for Multilingual Natural Language Processing*. PhD thesis, University of Melbourne, 2007.

Taylor L. Booth. Probabilistic representation of formal languages. In *Proceedings of the 10th Annual Symposium on Switching and Automata Theory*, pages 74–81, Waterloo, Canada, 1969.

Thorsten Brants. TnT - a statistical part-of-speech tagger. In *Proceedings of the 6th ACL Conference on Applied Natural Language Processing*, pages 224–231, Seattle, USA, 2000.

Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth Publishing Company, 1984.

Joan Bresnan and Ronald M. Kaplan. Lexical-Functional Grammar. A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, 1982.

Ulrich Callmeier. PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):99 – 108, 2000.

Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, 1992.

Bob Carpenter. Skeptical and credulous default unification with applications to templates and inheritance. In *Inheritance, defaults and the lexicon*, pages 13–37. Cambridge University Press, 1993.

John Carroll and Stephan Oepen. High-efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing*, pages 165 – 176. Jeju, Korea, 2005.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 132–139, Seattle, USA, 2000.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43th Meeting of the Association for Computational Linguistics*, pages 173–180, Ann Arbor, USA, 2005.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

Stephen Clark and James R. Curran. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 282–288, Geneva, Switzerland, 2004.

Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33 (4):493–552, 2007.

John Cocke. *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University, 1969.

Paul R. Cohen. *Empirical methods for artificial intelligence*. MIT Press, Cambridge, USA, 1995.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the Association for Computational Linguistics and the 7th Conference of the European Chapter of the ACL*, pages 16–23, Madrid, Spain, 1997.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42th Meeting of the Association for Computational Linguistics*, pages 111–118, Barcelona, Spain, 2004.

Ann Copestake. Defaults in lexical representation. In *Inheritance, defaults and the lexicon*, pages 223–245. Cambridge University Press, 1993.

Ann Copestake. *Implementing Typed Feature Structure Grammars*. CSLI Publications, 2002.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan Sag. Minimal recursion semantics: An introduction. *Research on Language & Computation*, 3(4):281–332, 2005.

Michael A. Covington. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, Athens, USA, 2001.

Bart Cramer. *Improving the feasibility of precision-oriented HPSG parsing*. PhD thesis, Saarland University, 2011.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42th Meeting of the Association for Computational Linguistics*, pages 423–429, Barcelona, Spain, 2004.

Walter Daelemans, Véronique Hoste, Fien De Meulder, and Bart Naudts. Combined optimization of feature selection and algorithm parameters in machine learning of language. In *Proceedings of the 14th European Conference on Machine Learning*, pages 84–95, Cavtat, Croatia, 2003.

Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51:107–113, 2008.

Yuan Ding and Martha Palmer. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43th Meeting of the Association for Computational Linguistics*, pages 541–548, Ann Arbor, USA, 2005.

Rebecca Dridan. *Using Lexical Statistics to Improve HPSG Parsing*. PhD thesis, Saarland University, 2009.

Rebecca Dridan and Stephan Oepen. Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Workshop on Parsing Technologies*, pages 225–230, Dublin, Ireland, 2011.

Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13 (2):94 – 102, 1970.

Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 973–978, Seattle, USA, 2001.

Johan Benum Evensberget. Context-free approximation of large unification-based grammars. Master's thesis, University of Oslo, 2011.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

Dan Flickinger. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6 (1):15 – 28, 2000.

Dan Flickinger, Stephan Oepen, and Gisle Ytrestøl. Wikiwoods: Syntacto-semantic annotation for English wikipedia. In *Proceedings of the 7th International Conference on Language Resources and Evaluation*, pages 1665–1671, Valletta, Malta, 2010.

Nelson W. Francis. A tagged corpus – problems and prospects. In Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik, editors, *Studies in English linguistics for Randolph Quirk*, pages 192–209. Longman Higher Education, 1979.

Nelson W. Francis and Henry Kučera. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, 1982.

Sanae Fujita, Francis Bond, Stephan Oepen, and Takaaki Tanaka. Exploiting semantic information for HPSG parse selection. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 25–32, Prague, Czech Republic, 2007.

R. Garside and N. Smith. A hybrid grammatical tagger: Claws4. In *Corpus Annotation: Linguistic Information from Computer Text Corpora*, pages 102–121. Longman, London, UK, 1997.

Jean Mark Gawron, Jonathan King, John Lamping, Egon Loebner, E. Anne Paulson, Geoffrey K. Pullum, Ivan A. Sag, and Thomas Wasow. Processing English with a generalized phrase structure grammar. In *Proceedings of the 20th Meeting of the Association for Computational Linguistics*, pages 74–81, Toronto, Canada, 1982.

Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, USA, 1985.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

Tadayoshi Hara, Yusuke Miyao, and Junichi Tsujii. Adapting a probabilistic disambiguation model of an HPSG parser to a new domain. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing*, pages 199–210. Jeju Island, Korea, 2005.

Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. In *Machine Learning*, volume 11, pages 63–91, 1993.

Ray S. Jackendoff. *The Architecture of the Language Faculty*. MIT Press, 1997.

Frederick Jelinek. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13(6):675–685, 1969.

Thorsten Joachims. Making large-scale support vector machine learning practical. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in kernel methods*, pages 169–184. Cambridge, USA, 1999.

Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, Philadelphia, USA, 2006.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77:27–59, 2009.

Aravind K. Joshi. An introduction to Tree Adjoining Grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87 – 115. John Benjamins, Amsterdam, The Netherlands, 1987.

Aravind K. Joshi and Bangalore Srinivas. Disambiguation of super parts of speech (or supertags): almost parsing. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 154–160, Kyoto, Japan, 1994.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, second edition, 2008.

Ronald M. Kaplan. *A general syntactic processor*. Natural Language Processing. Prentice Hall, 1973.

Ronald M. Kaplan and John T. Maxwell III. A grammar writers's cookbook. Technical report, Xerox PARC, 1996.

Tadao Kasami. An efficient recognition and syntax algorithm for context-free languages. Technical report, Air Force Cambrige Research Laboratory, Bedford, USA, 1965.

Martin Kay. Algorithm schemata and data structures in syntactic processing. In Barbara J. Grosz, Karen Sparck Jones, and Bonnie Lynn Weber, editors, *Readings in Natural Language Processing*, pages 35–70. Morgan Kaufmann, 1980.

Martin Kay. Guides and oracles for linear-time parsing. In *Proceedings of the 6th International Workshop on Parsing Technologies*, pages 6–9, Trento, Italy, 2000.

S. Sathiya Keerthi, S. Sundararajan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A sequential dual method for large scale multi-class linear SVMs. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 408–416, Las Vegas, USA, 2008. ACM.

Taku Kudo and Yuji Matsumoto. Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning*, pages 63–69, Taipei, Taiwan, 2002.

Jonathan K. Kummerfeld. Adaptive supertagging for faster parsing. Honours thesis, The University of Sydney, 2009.

Jonathan K. Kummerfeld, Jessika Roesner, and James R. Curran. Faster parsing and supertagging model estimation. In *Proceedings of the Australasian Language Technology Association Workshop*, pages 62–70, Sydney, Australia, 2009.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English. The Penn Treebank. *Computational Linguistics*, 19:313 – 330, 1993.

André F. T. Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. Stacking dependency parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166, Honolulu, USA, 2008.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1671–1676, Hyderabad, India, 2007.

Takuya Matsuzaki and Jun'ichi Tsujii. Comparative parser performance analysis across grammar frameworks through automatic tree conversion using synchronous grammars. In *Proceedings of the 22nd International Conference on Computational Linguistics*, volume 1, pages 545–552, Manchester, UK, 2008.

John T. Maxwell III and Ronald M. Kaplan. Unification-based parsers that automatically take advantage of context-freeness. In *Proceedings of the 1st LFG Conference*, Grenoble, France, 1996.

Kate McCarthy, Bibi Zabar, and Gary Weiss. Does cost-sensitive learning beat sampling for classifying rare classes? In *Proceedings of the 1st International Workshop on Utility-Based Data Mining*, pages 69–77, Chicago, USA, 2005.

David McClosky, Mihai Surdeanu, and Christopher Manning. Event extraction as dependency parsing. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics*, Portland, USA, 2011.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Haji. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, Canada, 2005.

Yusuke Miyao. Packing of feature structures for efficient unification of disjunctive feature structures. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics*, pages 579 – 584, College Park, USA, 1999.

Yusuke Miyao, Takaki Makino, Kentaro Torisawa, and Junichi Tsujii. The LiLFeS abstract machine and its evaluation with LinGO. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):47 – 62, 2000.

Yusuke Miyao, Takashi Ninomiya, and Junichi Tsujii. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In *Proceedings of the 1nd International Joint Conference on Natural Language Processing*, pages 684–693, Hainan Island, China, 2004.

Yusuke Miyao and Junichi Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43th Meeting of the Association for Computational Linguistics*, pages 83–90, Ann Arbor, USA, 2005.

Yusuke Miyao and Junichi Tsujii. Feature forest models for probabilistic hpsg parsing. *Computational Linguistics*, 34(1):35–80, 2008.

Takashi Ninomiya, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. A log-linear model with an n-gram reference distribution for accurate HPSG parsing. In *Proceedings of the 10th International Workshop on Parsing Technologies*, pages 60–68, 2007.

Takashi Ninomiya, Takuya Matsuzaki, Nobuyuki Shimizu, and Hiroshi Nakagawa. Deterministic shift-reduce parsing for unification-based grammars. *Natural Language Engineering*, 17(3):331–365, 2011.

Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Junichi Tsujii. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 155–163, Sydney, Australia, 2006.

Takashi Ninomiya, Nobuyuki Shimizu, Takuya Matsuzaki, and Hiroshi Nakagawa. Deterministic shift-reduce parsing for unification-based grammars by using default unification. In *Proceedings of the 12th Conference of the European Chapter of the ACL*, pages 603–611, Athens, Greece, 2009.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France, 2003.

Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together. (Workshop at ACL-2004)*, pages 50–57, Barcelona, Spain, 2004.

Joakim Nivre. Dependency grammar and dependency parsing. Technical report, Växsjö University, 2005.

Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the 47th Meeting of the Association for Computational Linguistics*, pages 351–359, Singapore, 2009.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Workshop on Parsing Technologies*, pages 73–76, Paris, France, 2009.

Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Meeting of the Association for Computational Linguistics: Human Language Technology*, pages 950–958, Columbus, USA, 2008.

Stephan Oepen and John Carroll. Ambiguity packing in constraint-based parsing. Practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 162 – 169, Seattle, USA, 2000.

Stephan Oepen and John Carroll. Efficient parsing for unification-based grammars. In Stephan Oepen, Daniel Flickinger, Junichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*, pages 195 – 225. CSLI Publications, Stanford, USA, 2002.

Stephan Oepen, Helge Dyvik, Jan Tore Lønning, Erik Velldal, Dorothee Beermann, John Carroll, Dan Flickinger, Lars Hellan, Janne Bondi Johannessen, and Paul Meurer. Som å kapp-ete med trollet? - towards MRS-based Norwegian-English machine translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 11–20, Baltimore, USA, 2004a.

Stephan Oepen, Daniel Flickinger, Kristina Toutanova, and Christopher D. Manning. LinGO Redwoods. A rich and dynamic treebank for HPSG. *Journal of Research on Language and Computation*, 2(4):575 – 596, 2004b.

Stephan Oepen, Daniel Flickinger, J. Tsujii, and Hans Uszkoreit, editors. *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*. CSLI Publications, Stanford, USA, 2002a.

Stephan Oepen and Jan Tore Lønning. Discriminant-based MRS banking. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 1250–1255, Genoa, Italy, 2006.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Chris Manning, Dan Flickinger, and Thorsten Brants. The LinGO Redwoods treebank. Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–5, 2002b.

Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Boston, USA, 1984.

Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. The University of Chicago Press and CSLI Publications, Chicago and Stanford, USA, 1994.

Keith Rayner and Lyn Frazier. Parsing temporarily ambiguous complements. *The Quarterly Journal of Experimental Psychology*, 39(4):657–673, 1987.

Jonathon Read, Dan Flickinger, Rebecca Dridan, Stephan Oepen, and Lilja Øvrelid. The WeSearch Corpus, Treebank and Treecache. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, Istanbul, Turkey, 2012.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, USA, third edition, 2009.

Ivan A. Sag, Timothy Baldwin, Francis Bond, Ann Copestake, and Dan Flickinger. Multiword expressions: A pain in the neck for NLP. In *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics*, pages 1–15, Mexico City, Mexico, 2002.

Kenji Sagae and Alon Lavie. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies*, pages 125–132, Vancouver, Canada, 2005.

Kenji Sagae and Alon Lavie. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, NAACL-Short '06, pages 129–132, New York, USA, 2006.

Ingo Schrøder. *Natural Language Parsing with Graded Constraints*. PhD thesis, Univeristy of Hamburg, 2002.

Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, USA, 2000.

György Szarvas, Veronika Vincze, Richárd Farkas, and János Csirik. The BioScope corpus: annotation for negation, uncertainty and their scope in biomedical texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing. (Workshop at ACL-2008)*, BioNLP '08, pages 38–45, Columbus, USA, 2008.

Lucien Tesnière. *Elements de syntaxe structurale*. Klincksieck, Paris, France, 1959.

Kentaro Torisawa and Junichi Tsujii. Computing phrasal signs in HPSG prior to parsing. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 949 – 955, Copenhagen, Denmark, 1996.

Kristina Toutanova, Christopher D. Manning, Stuart M. Shieber, Dan Flickinger, and Stephan Oepen. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories*, pages 253–263, Sozopol, Bulgaria, 2002.

Yoshimasa Tsuruoka, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou, and Junichi Tsujii. Developing a robust part-of-speech tagger for biomedical text. In *Advances in Informatics*, pages 382–392. Springer, Berlin, Germany, 2005.

Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne, Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, Klaus Netter, Günter Neumann, Stephan Oepen, and Stephen P. Spackman. DISCO — an HPSG-based NLP system and its application for appointment scheduling. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan, 1994.

Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer, New York, USA, 1995.

Erik Velldal, Lilja Øvrelid, and Stephan Oepen. Resolving speculation: MaxEnt cue classification and dependency-based scope rules. In *Proceedings of the 14th Conference on Computational Natural Language Learning (Shared Task)*, pages 48–55, Uppsala, Sweden, 2010.

Wolfgang Wahlster, editor. *Verbmobil. Foundations of Speech-to-Speech Translation*. Springer, Berlin, Germany, 2000.

Terry Winograd. Understanding Natural Language. *Cognitive Psychology*, 3(1):1–191, 1972.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206, Nancy, France, 2003.

David H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189 – 208, 1967.

Gisle Ytrestøl. Technical summary — selection and preprocessing of the WeScience Corpus. Technical report, University of Oslo, 2009. URL `http://www.delph-in.net/wescience/Ytrestol:09.pdf`.

Gisle Ytrestøl. CuteForce – Deep deterministic HPSG parsing. In *Proceedings of the 12th International Workshop on Parsing Technologies*, pages 186–198, Dublin, Ireland, 2011a.

Gisle Ytrestøl. Optimistic backtracking: a backtracking overlay for deterministic incremental parsing. In *Proceedings of the ACL 2011 Student Session*, HLT-SS '11, pages 58–63, Portland, USA, 2011b.

Gisle Ytrestøl, Dan Flickinger, and Stephan Oepen. Extracting and annotating Wikipedia sub-domains — Towards a new eScience community resource. In *Proceedings of the 7th Workshop on Treebanks and Linguistic Theories*, pages 185 – 197, Groningen, The Netherlands, 2009.

Daniel Zeman and Zdeněk Žabokrtský. Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the 9th International Workshop on Parsing Technologies*, Vancouver, Canada, 2005.

Yi Zhang, Rebecca Dridan, Hans-Ulrich Krieger, and Stephan Oepen. Robust parsing, meaning composition, and evaluation. Unpublished Manuscript, 2012.

Yi Zhang and Hans-Ulrich Krieger. Large-scale corpus-driven PCFG approximation of an HPSG. In *Proceedings of the 12th International Workshop on Parsing Technologies*, Dublin, Ireland, 2011.

Yi Zhang, Stephan Oepen, and John Carroll. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th International Workshop on Parsing Technologies*, pages 48–59, Prague, Czech Republic, 2007.

Yue Zhang and Stephen Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, 2008.

Yue Zhang and Stephen Clark. Shift-reduce CCG parsing. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics*, pages 683–692, Portland, USA, 2011a.

Yue Zhang and Stephen Clark. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151, 2011b.

Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics*, pages 188–193, Portland, USA, 2011.